RMIT UNIVERSITY

Secure Privacy-preserving Computing Applications on Cloud Using Homomorphic Cryptography

A thesis submitted in fulfilment of the requirements for the degree of Doctor of Philosophy

Vu Huy Mai

Bachelor of Computer Science, Honours, RMIT

School of Science

College of Science, Engineering and Health

RMIT University

August 2017

Declaration

I certify that except where due acknowledgement has been made, the work is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program; any editorial work, paid or unpaid, carried out by a third party is acknowledged; and, ethics procedures and guidelines have been followed.

I acknowledge the support I have received for my research through the provision of an Australian Government Research Training Program Scholarship.

Vu Huy Mai

9th, August, 2017

Acknowledgement

The thesis is finally finished! I am very happy that I am writing the acknowledgement, indicating that it marks the end of a very important chapter in my life.

Six years ago , when I just finished my bachelor degree in IT, I met Associate Professor Ibrahim Khalil to discuss potential research pathways for my graduate study. It has been a great conversation guiding my in the right direction in research until now. When I finished the thesis, I can say that I have made the right decision to do research in security and cryptography. My years as a graduate student have been the most rewarding years in my life. It was a true adventure; a job I greatly enjoyed, working with so many new technologies and technical areas to broaden my knowledge.

First of all, I would like to thank my first supervisor Associate Professor Ibrahim Khalil. I feel blessed to have been given the opportunity to learn from and work with him. He has provided me with a great deal of encouragement and support when I needed it the most. He also provided me with a new perspective on various topics of discussion. He always had time to read my papers, provide valuable comments and boost my spirits and kept me motivated throughout the duration of the PhD.

I would like to thank my family, especially my mother, who always helped me on everything and provided emotional strength often while being thousands of kilometres away.

Finally, I would like to thank all my friends in and outside of RMIT that directly or indirectly supported me throughout this time.

Credits

Portions of the material in this thesis have previously appeared in the following publications or to be submitted soon after the submission of this thesis:

- Mai V, Khalil I. Design and implementation of a secure cloud-based billing model for smart meters as an Internet of things using homomorphic cryptography. *Future Generation Computer Systems*. 2016 Jun 22.
  http://dx.doi.org/10.1016/j.future.2016.06.003

- Mai V, Khalil I. Cloud-based Homomorphic Secret Sharing and Access Control. To be submitted to the International Journal of Future Generation Computer Systems.

- Mai V, Khalil I. Secure Pricing Comparison Model Using Homomorphic Cryptography. To be submitted to the Special Issue on Security and Privacy in Cyber Physical Systems (Future Generation Computer Systems)

Abstract

The advancement of cloud computing technologies has provided users and business organisations with various cloud-based options to store and access information externally, across multiple platforms and geographic locations. The cloud also has the ability to deliver scalable and high-performance computing services on demand and in a cost-effective manner while helping users to avoid the trouble of maintaining large data centres and complex computing facilities. The economies of scale increase revenue for cloud providers and lower costs for cloud users. The resulting on-demand model of computing allows providers to achieve better resource utilization through statistical multiplexing, and enables users to avoid the costs of resource over-provisioning through dynamic scaling.

However, there are major security and privacy concerns when data is stored in external cloud storage systems. For example, when personal information is stored in unencrypted formats on the cloud, service providers can learn many details about the users such as their preferences, past behaviours and biometric identities. The widely distributed nature of cloud architectures means that server farms can be located in many countries or geographic locations that might be under different laws and regulations regarding user privacy. Furthermore, cloud service providers may encrypt data in-transit, but not while user data is stored on their servers, causing the reluctance of many business organisations to outsource the storage of their sensitive and valuable data, which can be major targets for attacks coming from both outside attackers and insiders.

Therefore, encrypting the data when it is stored on the cloud is an important task to guarantee the confidentiality and privacy of users data. However, traditional cryptographic techniques make it difficult for processing tasks such as searching, updating or checking the integrity of encrypted data without asking clients to download and decrypt large amounts of data from the cloud. To realise the full potential of cloud computing, better cryptographic schemes are required. They should enable the cloud to perform various computing operations

on encrypted data and return encrypted results to customers. Another desirable feature is how a cryptographic scheme can allow different parties to combine their encrypted data and perform some computing tasks on the cloud without compromising the confidentiality and privacy of the data of each party.

Recently, homomorphic cryptography has increasingly been the focus of researchers because this technology has a great potential to provide the desirable features described above. Homomorphic encryption can be implemented either as a symmetric or a public-private asymmetric key paradigm. This technique allows many types of computing operations to be performed on ciphertext and output encrypted results which, when decrypted, are found to be identical to the results of the same operations performed on plaintext data. With a homomorphic cryptosystem, many computational circuits can now be homomorphically evaluated, producing programs that might be run on encryptions of their inputs to produce an encryption of their output. Since the inputs of such programs are encrypted, a computation task can be performed on an untrusted cloud without revealing any inputs and internal states.

In this thesis, we focus on the design and implementation of various application models of homomorphic cryptography so that the cloud can be used more effectively and securely to store and process sensitive customer data. Our research works throughout many chapters of this thesis also provide valuable information regarding the security of homomorphic cryptography in many use case scenarios. We illustrate how homomorphic cryptography can be applied effectively with all of its flexibility, power and usefulness in many applications ranging from smart grid, e-commerce to secret sharing. In this thesis, we also propose approaches to enhance the efficiency and effectiveness of homomorphic cryptography, so that these cryptographic schemes can be applied not only in current cloud-based application, but also in larger, more mission-critical applications in the future.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In recent years, there have been many major advancements in cloud computing technologies, making it an efficient computational paradigm to provide users with high quality, customized and reliable services at reduced cost. Many applications and databases have been moved to the cloud or completely redesigned and implemented to work on large, centralized data centres. Cloud service providers also play a significant roles in redefining the current methods that software and hardware are designed, implemented or purchased. They can provide users and customers around the world with flexible options to purchase on demand storage and computational resources [Armbrust et al., 2010]. As a consequence, business users no longer need to acquire and manage large, complicated and often costly computing infrastructures. These benefits of the cloud is very attractive, especially for start-up businesses and small companies because they only need to pay only for the resources they actually use, thereby reducing operational costs through economies of scale.

Although cloud computing has brought many benefits to its users, there are still major challenges that have to be addressed before cloud platforms can be widely used with reliability. The most outstanding among those challenges are the cloud-based data confidentiality, user privacy and the general security of the platforms [Takabi et al., 2010]. While traditional computing models can provide users with full control of their data storage and computation, this is often not the case when a cloud service is used because most of the physical data and computing infrastructures are managed by cloud service providers. Hence, there is a strong possibility for security breaches to happen when data owners do not have sufficient tools or effective mechanism to control the security of their cloud-based data. It often means that users have no choice but to rely on protection mechanisms offered by cloud service providers.

However, the distributed nature of the cloud means that cloud servers can be stored all over the world in many places having different rules and regulations regarding user privacy and data security.

One very effective method which has been used to provide data security is using cryptography [Patwal and Mittal, 2014]. Historically, cryptography was used to send secret messages through insecure medium with well-known cryptographic schemes such as Caesar Cipher or Hill Cipher. As time goes on, cryptography continues to play a more important role than ever in many computational models, especially when there is a large amount of sensitive data sent and received all over the Internet. Furthermore, many symmetric and asymmetric key encryption schemes have also been applied to secure date stored on the cloud. Data owners will typically encrypt their data before outsourcing the ciphertexts to the cloud. Only selected users possessing the decryption keys are able to download, decrypt and retrieve the plaintext for further processing. Although this approach has offered high levels of security, there is a limitation. Many of the data processing tasks must still be performed on the client side, requiring client computers to have significant computing power, especially when there is a large volume of data to be processed. This limitation is likely to become a major disadvantage of cloud-based models that only store encrypted data, when the client side has limited computing resource such as when a mobile phone is used or when there is a large amount of data to be processed.

Simply storing encrypted data on the cloud makes cloud computing less attractive because not many computational operations can be performed when the data is encrypted. The cloud just stores data in encrypted formats and manages access so that only authorised party can download the encrypted data. Therefore, more effective and efficient cloud computing models are required to provide the ability to compute on encrypted data in addition to storing data securely. Such computational models must provide authorised parties with the capability to request the cloud to perform many computing operations securely on encrypted data. The confidentiality and privacy of the data are maintained throughout these computing processes because no decryption will be required at any stage and the outputs will also be in encrypted forms which can be decrypted only by authorised parties.

In this thesis, we will present many novel privacy-preserving cloud-based computation models which allow users to store and process encrypted data securely on the cloud. This feature is achieved through our application of homomorphic cryptography, which not only encrypts data but also provides a mathematical basis for various computing operations to be performed directly on encrypted data. In each core chapter presented in this thesis, we demonstrate

the usefulness and effectiveness of homomorphic cryptography in protecting user privacy and confidentiality of cloud-based data by designing, analysing and implementing various practical application models with the cloud as a centralised computing hub. However, before describing the details of those models, we will explore a few important aspects of privacy in business computing as well as how a privacy-preserving cloud-based computation model is constructed.

1.1 Security, Privacy-preserving Protocols on Cloud

1.1.1 Security and Privacy in Cloud Computing

Privacy is a very important concept in everyday life. It can be understood as the ability of a group or an individual to choose how much information to share or keep to themselves. Although what is regarded as private can be understood differently among cultures and individuals, a common rule is that there always exists sensitive and valuable information which a person or a business organisation wants to protect or strictly control access to. Privacy requirements are even more significant in the world of cloud computing in which it is the data stored on the cloud that owners want to protect or control disclosure to others. A privacy breach of cloud-based data often has severe and immediate consequences affecting many customers of a business organisation.

Data security is commonly referred to as the confidentiality, availability, and integrity of data. In other words, it is all of the practices and processes that are in place to ensure data isn't being used or accessed by unauthorized individuals or parties. Data security ensures that the data is accurate and reliable and is available when those with authorized access need it. A data security plan includes facets such as collecting only the required information, keeping it safe, and destroying any information that is no longer needed. These steps will help any business meet the legal obligations of possessing sensitive data.

Companies need to enact a data security policy for the sole purpose of ensuring data privacy or the privacy of their consumers' information. More so, companies must ensure data privacy because the information is an asset to the company. A data security policy is simply the means to the desired end, which is data privacy. However, no data security policy can overcome the willing sell or soliciting of the consumer data that was entrusted to an organization.

Privacy of information stored on the cloud can also be better understood from examples of cloud-based applications in different areas of our society such as healthcare, business, social networking and education. Following are short descriptions of those areas and how the privacy

of digital data stored on cloud is often addressed:

- **Security and privacy of electronic medical records.** Healthcare information is one of the most important types of data that need to be protected. Patients going to a hospital often reveal important details regarding their well-being to medical professionals for diagnosis and treatments. These electronic medical records can be stored and managed by a hospital or third-party cloud services especially when there is a large volume of data. To protect the privacy of patients, there are many rules and regulations such as the Health Insurance Portability and Accountability Act (HIPAA) specifying how healthcare professionals such as doctors and nurses can obtain access to those data with direct permissions from data owners. Therefore, when the data is stored on cloud, there must be technical means to achieve those privacy requirements.

- **Security and privacy of government data.** Many governments around the world often create web portals where their citizen can quickly and conveniently get access to many governmental services such as social benefits, healthcare protection plans, job services, etc. To achieve such functionalities, governments often collect a lot of information from their citizens. This huge amount of data also need to be stored and managed effectively and securely to minimise the chance of a security breach which can compromise the privacy of many citizens. There are laws regarding such protection of privacy as well as the need for better technical solutions to address those challenges.

- **Security and privacy of corporate data.** For many business organisations, data is the most important part of their operations. There have always been the need to collect more and more data as well as how to store and control access to those data so that the maximum benefit can be achieve while minimising the risk of an information leakage. The privacy of the business data managed by those companies are inherently related to the privacy of their customers. Therefore, while many cloud-based data storage and processing solutions are very attractive due to their scalability and potential to save a significant amount of operational cost, business organisations are facing the risk of losing revenue and the trust of their customers to the point of shutting down if they do not provide solid privacy protection schemes.

- **Security and privacy of social network data.** Social networks have an ever significant role in the digital age with prominent websites such as Facebook, Twitter, Instagram, Youtube, Flickr, etc. having millions of users communicating and sharing information

with each other. Despite the capability to provide quick and easy methods to establish connections among people, many social networking sites still lack effective mechanisms to protect the privacy of their users or even exploit those sensitive data for other purposes. Furthermore, the users may not have enough knowledge about their online privacy or are provided with insufficient technical means to control their own privacy.

In the previous paragraphs, we have defined privacy in cloud computing and described how this important concept can be understood from different contexts such as e-health, government, corporate, and social network data. One important conclusion which can be made from this investigation is that protecting user privacy and data confidentiality is one of the most important tasks having direct influence on the success of many cloud computing models. Hence, there is a need for privacy-preserving protocols to not only provide protection to cloud-based data by means of encryption but also enhance a cloud-computing model by having the capability to computing on encrypted data. In the next section, we will describe the common characteristics of such privacy-preserving protocols as well as define the basic security models often found in many research works in this area.

1.1.2 Basic Features of Secure and Privacy-preserving Protocols

Various literature reviews in the area of cloud computing have shown that security, together with interoperability and portability are the three main hurdles preventing a more widespread adoption of cloud solutions. From a security perspective, cloud computing has increased the scale and complexity of many trust and privacy issues. Storing data on cloud also introduces new problems requiring urgent solutions to improve the confidence of cloud-based users. A solution in this context is often presented as a protocol in which the author proposes a cloud computing model and various methods to securely interact with cloud-based data. All of these models and protocols have some basic characteristics which we will describe in the following paragraphs.

A common feature in many privacy-preserving protocol is trust. A cloud computing model must provide mechanisms for its users to establish a solid trust so that they have full confidence when outsourcing their data on the cloud. Cloud providers also need authentication and authorisation methods to trust their customers with access to the services. Without adequate trust establishment mechanisms, malicious users can get unauthorised access to vast amount of data stored on the cloud, causing major damages to a cloud-based business. Therefore,

all cloud computing model provides strong authentication and authorisation mechanisms to validate user credentials as well as control access to cloud-based data.

Aside from establishing and managing trust in cloud-based computation models, another important aspect is security. While users can have full control over data storage and computation using traditional computing models, there are certain limitations cloud users will encounter when outsourcing their data to the cloud due to the lack of direct control over how their data is stored and processed. Cloud security can be classified into two major classes which are storage security and computation security. Any strongly secure cloud computing model will have to ensure both aspects. Storage security means maintaining the integrity of user data stored on third-party cloud servers. This can be achieved by using a combination of cryptographic keys, hash functions and various digital signature algorithms. While there are many proposed research works focus on storage security, this feature alone is not enough to create an effective cloud-based computational model. Another important feature is computation security which is about ensuring the correctness and confidentiality of all types of computing operations executed on cloud servers.

A typical approach to achieve computational security is using a trusted third party to collect data inputs from owners and perform all required computing operations before returning the results to the owners. Privacy is ensured because the trusted entity is responsible for protecting the data from unauthorised access. This approach, however, should not be applied to cloud computing models because cloud servers cannot be fully trusted. Furthermore, the trusted server can easily be the focus of attacks causing major security breaches especially when data is not stored in encrypted forms. Therefore, a common expectations of all cloud-based data management models is that data is encrypted in storage. In this case, it is essential to make sure that computation is performed on encrypted data to produce encrypted results which can only be decrypted by authorised parties having correct decryption keys.

Another common features of all privacy-preserving protocols is that each of them will have to provide details about the privacy model it will be using. The one that has been used in many protocols is the semi-honest (or honest, but curious) model. Protocols assumed the use of this semi-honest model often allow their participants to monitor and record all transaction details or any other data during the execution steps of such protocols. When a protocol finishes running, participants can perform further analysis of the data they have collected to gain additional information rather than just the inputs and outputs they are supposed to get. Another privacy model which has often been used is the malicious model. This is an adversarial model which is often used to describe situations when one or more participants do not act according to the

predefined set of rules given by a protocol. The set of possible actions in the malicious model is defined very broadly, including all actions in the semi-honest model together with more disruptive behaviours such as skipping immediate steps or aborting the protocol altogether. Therefore, many security and privacy-preserving protocols are often proven to be valid under the semi-honest model first and then their proofs will be generalized to show how such protocols are secure under the malicious model. These proofs are very complicated, requiring the use of complex mathematical tools such as complexity theory, one-way functions and zero-knowledge proof.

1.2 Motivations and Problem Statements

In the previous section, we have defined privacy in different aspects of cloud computing as well as described all common features of privacy-preserving protocols. The most important task in determining the success of a cloud computing model is providing strong security of both data stored on the cloud and all computing operations performed over the encrypted data. This task also requires the protection of user privacy at the lowest levels during the execution stage of a protocol. Following the given definitions and requirements, this thesis will propose an implement of various cloud-based privacy-preserving computing models in different realistic scenarios. There is an important question that this thesis will answer: how to design and implement cloud-based computing models which can both be used for storage and computation of outsourced data as well as protecting data security and preserving the privacy of the users. This thesis will show how to answer that question in three different cloud-computing application scenarios:

- **Problem 1: How to design and implement a secure model for smart meters data processed on semi-trusted cloud servers.** A smart grid may introduce many outstanding security and privacy issues, especially when smart meters are connected to public networks in which not only personally identifiable information but also energy consumption data relating to behaviours and movements of users is frequently exchanged and processed in large volumes. The security and privacy issues outlined above have prevented the cloud from reaching its full potential in supporting a smart grid, especially when many components of the grid are connected as an Internet of things via the public network. Smart meters are constrained devices having limited capacity and incapable of performing complex computing tasks on energy usage data. A smart grid needs a

cloud computing system with its unlimited storage and processing capabilities to support complex computing tasks such as billings and analysing data. Therefore, we want to propose a cloud-based data storage and processing model applicable for a smart grid which will utilise the full strength of cloud computing and preserves user privacy and the confidentiality of data exchanged on the grid.

- **Problem 2: How to create a secure cloud-based pricing comparison model with privacy protection features.** Trust is a serious matter in cloud computing as described in the previous section. There are scenarios in which the parties performing the computing operations can not be trusted. Specifically, there are cases where the processing entity is semi-honest or even malicious. In the former case, the computing entity is required to follow the protocol, but are permitted to record all data collected during the execution of the protocol. At the end of a computing session, the entity can analyse the data collected in an attempt to extract more information than just their inputs and the output of the function. If the computing entity is malicious, many unexpected behaviours can happen when the computing protocol is executed. For example, aborting the protocol can be used as a technique to attempt to learn the inputs and tamper with the final results. A practical example of these scenarios is in e-commerce when a client, an ordinary Internet user, who wants to participate in an e-commerce transaction but does not trust the e-commerce cloud server. Therefore, we want to create a privacy-preserving protocol which will process personal details of clients in a secure manner, allowing essential operations such as checking if the client has sufficient fund before approving a purchase while protecting data security and privacy of clients.

- **Problem 3: How to design and implement a confidential multi-party secret sharing and access control model on the cloud.** In large business organisations, the access to highly sensitive data is usually controlled by a group of users rather than a single user. In this case, members of the group controlling access to those data must have a mechanism to jointly authorise access to encrypted data stored on the cloud. Cryptographic keys can be mathematically divided into multiple independent pieces which are delivered to each member of the group controlling access. However, this scheme requires a trusted party to collect and combine shares of the encryption key before performing computation works to recover the original key. This requirement may not be applicable in cloud-based storage and computing scenarios in which the main computing agents are semi-trusted cloud servers located outside the business organisation. Therefore, a new

secure secret sharing model is required to protect the shares while computing and recovering the secret to preserve the privacy of each member of the group controlling access to the encrypted data.

- **Problem 4: How to propose a confidential e-commerce transaction model with one-time payment processing capability** For an e-commerce application to be successful, it must have the capability to secure personal and financial information of all its participants. Not only the data must be secured when stored on servers, but the integrity of these data must also be guaranteed in transit, so that any financial information which has already been used cannot be replayed in another transaction without the permissions of data owners. Recent incidents involving major data breaches in large e-commerce organisations have proven that potential privacy threats can severely impact the rapid development of electronic commerce. Therefore, in this research, we want to apply homomorphic cryptography to address these security challenges. We aim to propose an e-commerce model with strong privacy preserving features which are built using homomorphic cryptography. We also need to address the challenges of how to provide a novel one-time mechanism allowing all authentication details of a business transaction to be non-reusable immediately after they have been used for the first time. This important feature enables users of our model to purchase goods securely from an online merchant while guaranteeing the authentication and integrity of data.

1.3 Research Questions

In order to overcome the afore-mentioned research problems, the following research questions are defined with the aim of identifying the major objectives of our research as well as proposing effective and efficient solutions using homomorphic cryptography. In the subsequent parts of the thesis, we propose various working models to answer these questions and discuss all related aspects, including future research directions that can be followed based on these research questions. These questions also work as a benchmark to determine the viability of our proposed models. Following are the research questions we focus on in this thesis:

- First: How can homomorphic encryption be applied to secure the storage and computation of data on the cloud while ensuring the correctness of computing results and the confidentiality of user data?

- Second: How to design and implement new computing models that allow multiple users to contribute their data in encrypted formats to build secure cloud-based applications using homomorphic encryption as the main mechanism to compute on encrypted data?
- Third: How to apply the proposed models and assess its efficiency and effectiveness in practical application scenarios such as smart grid, secure e-commerce as well as secret sharing and access control?

1.4 Research Outcomes

With all the research problems identified and defined clearly in the previous section, we now present an overview of the research outcomes of this dissertation. To achieve comprehensive results for all problems listed above, we went through a formal and detailed process which comprises of many steps. Firstly, we tried to understand and analyse the requirements and contexts, ensuring that the problem domains were comprehended from the right perspectives. Then we performed extensive literature reviews so that our research works could inherit and benefit from the latest developments in the research areas of interest. The models that we proposed were carefully designed, implemented and experimented together with detailed security analyses for all models to prove their applicability and solidity in practical scenarios. Therefore, in this section, we can only provide a very brief summary of what we have achieved. For more details and in-depth discussions of these outcomes, readers should consult the core chapters of this dissertation.

- **Outcome 1: Design and implement a secure model for smart meters data processed on semi-trusted cloud servers.** Using homomorphic cryptography, we have successfully designed and implemented a model which allows the cloud to securely compute the aggregated energy consumption of a customer in a given period of time by performing homomorphic addition operation directly on an arbitrary number of encrypted fine-grained readings sent from the smart meter of that customer. The cloud does not need to perform any decryption during this computation process. Therefore, user privacy and data confidentiality are protected. We also propose a parallel version of our billing algorithm to utilise the processing capacity of multi-core processors in cloud servers, reducing computation time by about 50 percent compared to using our sequential homomorphic addition algorithm.

- **Outcome 2: Create a secure cloud-based pricing comparison model with privacy protection features.** We propose a new mechanism based on homomorphic encryption to minimise the use of a trusted third party and allow a client and server to validate e-commerce transaction outcomes in a reliable and secure manner. Specifically, our model allows homomorphic addition, comparison and integrity checking of comparison results to be performed over encrypted binary digits. Data security and client privacy are ensured because most of the computing operations are performed on encrypted data. The implementation of our of model is efficient and powerful because the cryptographic scheme we use has small ciphertext and key size with all homomorphic computing operations performed directly on encrypted binary numbers. Our computing model is scalable and can easily be extended to allow an arbitrary number of parties to join the homomorphic cloud-based computation.

- **Outcome 3: Propose a confidential multi-party secret sharing and access control model on the cloud.** The main focus of our model is the homomorphic secret sharing scheme to help any group of users in a business organisation to create a shared secret with most of the computing tasks to create, distribute and regenerate the shared secret to be done on the cloud. Users in the group are able to keep their own shares of the secret private while still being able to combine these shares on the cloud. We show how the cloud can do most of the computing task in the process of generating the shared secret without affecting data security and privacy of any user in the group. We also implement an access control model which applies the proposed homomorphic secret sharing scheme to provide a group of users with the ability to control access to encrypted documents stored on the cloud. Most of the computing operations required for each group member to authorise access are done on the cloud over encrypted shares of the secret, therefore the power of the cloud can be fully utilised without affecting the security of all shared secrets of the controlling party as well as the security of encrypted data stored on the cloud.

- **Outcome 4: Design a one-time e-commerce payment processing model using homomorphic cryptography.** In this research work, we propose a detailed design of a secure online payment processing model with all the major components required to protect sensitive customer financial information. An outstanding feature of our model is the separation of transaction information into encrypted packages only accessible to relevant parties involved in a transaction. The robustness of our design is ensured due

to the use of homomorphic cryptography when encrypting data packages and processing payment information directly in encrypted format. The non-reusability of encrypted data in our model is also guaranteed by the use of random factors such as unique time-stamps and random session keys, making sensitive information sent out by customer obsolete after the first use. This is an important mechanism to guard against many security risks such as a replay attack or an identity theft. Our model also have the ability to perform reliably in many different adversarial scenarios as proven in our work.

1.5 Contributions

Through the research works described in this thesis, we have made significant contributions in the area of security and privacy of cloud-based computational model. These contributions can be summarised as followed:

- **Privacy-preserving cloud-based computing models with homomorphic computational functions.** We have applied homomorphic cryptography in many application scenarios to create various practical cloud computing models with the capability to compute on encrypted data. Our models ensure that user privacy and data confidentiality are protected by means of cryptography because encrypted data is stored and processed mostly on the cloud using the public homomorphic key, hence no decryption is required.

- **Secure multi-party secret sharing on cloud using homomorphic cryptography.** We propose a key management scheme and access control model using homomorphic cryptography to allow the key generation work to be executed on semi-trusted cloud servers, opening new possibilities for applications that can fully utilise the computing power of the cloud while providing maximum protection for data security and user privacy.

- **Accurate and efficient implementations of privacy-preserving protocols.** We demonstrate in this thesis the applicability, effectiveness and efficiency of our computational models in real-world scenarios with practical data. We implement working applications to show that our theoretical models can be realisable in practice with different methods to measure the performance of our applications.

Chapter 2

Preliminaries

At the beginning of this chapter we will provide a broad overview of the computational problems that are related to homomorphic cryptosystems. We provide essential information to understand homomorphic cryptography and some of the approaches to this cryptographic paradigm. Then we make a survey of some well known homomorphic cryptosystems, generally in the order they were discovered as well as how these schemes show the homomorphic properties. Next, the basic characteristics of secure cloud-based applications are described with highlights of the major security threats and requirements about how to secure those cloud-based applications. Finally, we will provide an overview of some solutions that have been introduced in the literature to protect the security of those cloud-based applications.

2.1 Foundations of Homomorphic Cryptosystems

In this section, we present the foundations on which homomorphic cryptosystems are built. To show that a cryptosystem is robust, we need to demonstrate that if the encryption function can be inverted or broken its semantic security by an adversary, then the adversary will have no choice but to break some instance of an intractable problem. Therefore, a cryptosystem is only secure when it is based on an intractable problem. All the problems we present in this section will have the associated assumption that if the right parameters are chosen, each problem will be intractable. In the following definition, we describe how a computational problem is polynomially reducible to another computational problem and how two problems are computationally equivalent.

Definition 2.1. Let A and B be two computational problems. A is said to be polynomially

reducible to B if there exists an algorithm that solves A, which uses, as a subroutine, a hypothetical algorithm for solving B, and which runs in polynomial time if the algorithm for solving B does. If A is polytime reducible to B, and B is polytime reducible to A, then A and B are said to be computationally equivalent.

The hypothetical algorithm used for solving a computational problem is often referred to as an oracle, and works like a black box that, given an instance of the computational problem, returns a solution in polynomial time with non-negligible probability.

2.1.1 The Factoring Problem

The fundamental theorem of arithmetic states that every integer $n \geq 2$ can be expressed uniquely as a product of prime factors, i.e.,

$$n = {p_1}^{e_1} {p_2}^{e_2} \dots {p_k}^{e_k}$$

where the p_i's are distinct primes. The problem of determining the prime factorization of n, particularly when n is large and has few prime factors (typically two or three), forms the basis of security for many cryptosystems.

Definition 2.2. Given an integer n, the factoring problem is the problem of calculating the p_i's such that

$$n = {p_1}^{e_1} {p_2}^{e_2} \dots {p_k}^{e_k}$$

where each p_i is a prime and $p_i \neq p_j$ when $i \neq j$

Currently, the best known methods for factoring are the number field sieve [Lenstra et al., 1993] and the elliptic curve method [Lenstra Jr, 1987]. In general, the number field sieve is the most efficient algorithm for factoring n, assuming n has no small prime factors. The running time of the number field sieve depends on the bit-length of n, while the elliptic curve method depends on the bit-length of the smallest factor of n. In cryptographic applications, n is usually selected such that each of its prime factors are approximately the same bit-length. While $n = pq$ for primes p and q is commonly used, some cryptosystem use $n = p^2q$. The number field sieve remains the most efficient method of factoring numbers of this form; however, some elliptic curve factoring approaches specific to $n = p^2q$ have been studied, such as the approach given by Peralta and Okamoto [Peralta and Okamoto, 1996].

The difficulty of factoring n can be parameterized by ε, the bit-length of n, with the assumption that each prime factor of n is approximately the same size. To remain secure against modern attacks, it is generally regarded that ε should be at least 1024, if not 2048.

2.1.2 RSA / e'th Root Problem

The RSA encryption function is $c = m^e \bmod n$ where $m \in \mathbb{Z}_n$ and $n = pq$ for distinct odd primes such that the factoring problem is hard on n. Decrypting the ciphertext c is equivalent to taking the e'th root of c mod n. When the factorization of n, the private key, is unknown, this task should be intractable.

Definition 2.3. Let $n = pq$ for distinct odd primes p and q. Given integers $c, e \in \mathbb{Z}_n$ with $e > 2$, the RSA Problem is the problem of finding m such that $c = m^e \bmod n$, without knowledge of p and q.

For efficiency reasons, it is often desirable to choose e to be a small integer, such as $e = 3$, or any other special form that makes encryption more efficient. Thus, the RSA assumption is often stated in a stronger form.

Definition 2.4. Let $n = pq$ for distinct odd primes p and q. Given $c \in \mathbb{Z}_n$, the Strong RSA Problem is the problem of finding $m, e > 1$ such that $c = m^e \bmod n$, without knowledge of p and q.

An overview of the RSA and Strong RSA problems has been given by Rivest and Kaliski [Rivest and Kaliski, 2005].

A slightly different form of the Strong RSA Problem is also used in some cryptosystems. Let λ be the Carmichael Function, such that $\lambda(n) = m$, the smallest positive integer such that $a^m \equiv 1 \bmod n$ for all a relatively prime to n. The Carmichael function represents the exponent of the group $\mathbb{Z}_n^*$, and can be calculated by:

$$\lambda(n) = \begin{cases} \phi(n) \text{ for } n = p^\alpha, \text{ with } p = 2 \text{ and } \alpha \leq 2, \text{ or } p \geq 3 \\ \frac{1}{2}\phi(n) \text{ for } n = 2^\alpha \text{ and } \alpha \geq 3 \\ lcm(\lambda({p_i}^{\alpha_i})) \text{ for } n = \Pi_i {p_i}^{\alpha_i} \end{cases}$$

The problem of computing e'th roots when $\gcd(e, \lambda(n^2)) = 1$ and $n = pq$ is called the Computational Small e'th Roots Problem, and was posed by Catalano, Gennaro, Howgrave-Graham, and Nguyen [Catalano et al., 2001].

Definition 2.5. Let $n = pq$ for distinct odd primes p and q. Given $e \in \mathbb{Z}^2_{n^2}$ such that $\gcd(e, \lambda(n^2)) = 1$, and an integer y, the Computational Small e'th Roots Problem is the problem of computing x such that $x^e = y \bmod n^2$. The Decisional Small e'th Roots Problem is the problem of deciding whether or not a given x is an e'th root modulo n^2.

Each of the e'th root problems can be parameterized by ϵ, the bit-length of n.

2.1.3 Quadratic Residuosity Problem

The problem of deciding whether or not an integer is a quadratic residue modulo n forms the basis of security for the Goldwasser-Micali cryptosystem. Further variations of residue problems also form the basis for most modern homomorphic cryptosystems.

Definition 2.6. An integer a is said to be a quadratic residue modulo n if there exists $0 < x < n$ such that

$$x^2 \equiv a \bmod n$$

Otherwise, a is said to be a non-quadratic residue modulo n.

If n is an odd prime, then determining whether or not an integer a is a quadratic residue modulo p is equivalent to calculating the Legendre symbol

$$\left(\frac{a}{p}\right) = \begin{cases} 0 \text{ if } a \equiv 0 \bmod p \\ 1 \text{ if } a \neq 0 \text{ and there exists } x \in \mathbb{Z} \text{ such that } a \equiv x^2 \bmod n \\ -1 \text{ if no such } x \text{ exists} \end{cases}$$

which can be efficiently calculated by the formula

$$\left(\frac{a}{p}\right) = a^{\frac{p-1}{2}} \bmod p$$

For an odd composite number n, the Jacobi symbol can also be calculated

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right)^{e_1} \left(\frac{a}{p_2}\right)^{e_2} \cdots \left(\frac{a}{p_k}\right)^{e_k}$$

where $n = {p_1}^{e_1} {p_2}^{e_2} \ldots {p_k}^{e_k}$ is the prime factorization of n. Unlike the Legendre symbol, $\left(\frac{a}{n}\right) = 1$ does not imply a is a quadratic residue modulo n.

Definition 2.7. Let the set $\Re_n$ denote the set of quadratic residues modulo n. Given an odd composite integer $n > 3$ and an integer a such that $\left(\frac{a}{n}\right) = 1$, the problem of determining if $a \in \Re_n$ is called the Quadratic Residue Problem.

If $n = pq$ for distinct odd primes p and q, then $a \in \Re_n$ if and only if $\left(\frac{a}{p}\right) = 1$ and $\left(\frac{a}{q}\right) = 1$. Thus, knowledge of the factorization of n is sufficient to solve the quadratic residuosity problem. It is currently believed, although unproved, that solving the quadratic residuosity problem is equivalent to factoring. Assuming the truth of this conjecture, the security parameter for the quadratic residuosity problem is the same as the security parameter for the factoring problem.

2.1.4 The p-Subgroup Problem

Let p, q be distinct primes such that $p \nmid q - 1$, and consider the group $\mathbb{Z}^*_{p^2q}$. The order of $\mathbb{Z}^*_{p^2q}$ is $\phi(p^2q) = p(p-1)(q-1)$. By Sylows First Theorem [Gallian, 2009], $\mathbb{Z}^*_{p^2q}$ has exactly one subgroup of order p, which is the Sylow p-subgroup. Deciding if an element $a \neq 1 \in \mathbb{Z}^*_{p^2q}$ belongs to the unique subgroup of order p can be verified by checking if $a^p \equiv 1 \bmod p^2q$; i.e. the order of a in $\mathbb{Z}^*_{p^2q}$ is a divisor of p. Because p is prime, this can only occur if $|a| = 1$, which, by assumption, cannot occur, or $|a| = p$.

Verifying if an element belongs to the subgroup of order p is similar to the residue problems presented in the previous section, and it is conjectured to be difficult when then factorization of p^2q is unknown.

Definition 2.8. Let $n = p^2q$ for distinct odd primes p and q such that $p \nmid q - 1$. Then the problem of deciding whether or not a random element $a \neq 1 \in \mathbb{Z}^*_n$ is in the unique Sylow p-subgroup when the factorization of n is unknown is called the p-subgroup Problem.

As with the residuosity problems, the best known approach to solving the p-subgroup problem is through factoring $n = p^2q$. Thus, the p-subgroup problem can be parameterized by ϵ, the bit-length of n.

2.1.5 Discrete Logarithms and the Diffie-Hellman Problem

The discrete logarithm is the group equivalent of the logarithm function for real numbers.

Definition 2.9. Given a generator $\alpha \in \mathbb{G}$ of a cyclic group of order n, and given some $\beta = a^x \in \mathbb{G}$, the discrete logarithm of β in base α is the unique value $x \bmod n$. Given β, the problem of finding x is called the Discrete Logarithm Problem (DLP).

The difficulty of solving DLP depends on the representation of the group considered. The most general setting is the generic group model, in which no assumptions are made about the underlying structure of the group. Shoup has shown that the time needed to solve DLP in the

generic group model is at least $\Omega(\sqrt{n})$, where n is the size of the group. Solutions to DLP that have expected running time $O(\sqrt{n})$, exist, showing this bound as tight. Such approaches include Pollards rho algorithm, Pollards lambda algorithm, and the baby-step giant-step algorithm [lyn Teske, 2001].

A different approach, called the index-calculus method, does utilize the underlying group representation. If group elements can be efficiently represented using a factor base of small factors, such as integers whose prime factorizations contain only small prime powers, then the properties of the logarithm function can be used to reconstruct the logarithm of g from the logarithm of each of gs factors. In general, the index-calculus method is more efficient than the generic square-root methods, so for cryptographic purposes groups are usually chosen such that the index calculus method does not work.

As seen earlier, many definitions of security for cryptographic systems rely on the adversarys inability to distinguish between ciphertexts. When building cryptosystems that rely the difficulty of solving DLP, this often translates to the adversary being unable to distinguish between the discrete logarithms of two group elements. The Computational Diffie-Hellman Problem (CDH) and Decisional Diffie-Hellman Problem (DDH) attempt to capture the difficulty of problems related to DLP as they naturally arise in cryptography.

Definition 2.10. Given a generator $\alpha \in \mathbb{G}$, a cyclic group of order q, and two group elements α^a and α^b, the Computational Diffie-Hellman Problem (CDH) is the problem of calculating α^{ab} from α^a and α^b, without knowledge of a and b.

Clearly, a solution to DLP can be polynomially reduced to a solution to CDH, although it is currently unknown if a reduction in the other direction exists.

The decisional version of CDH requires the adversary to distinguish between α^{ab} and α^c for some random integer c, a situation that arises naturally from the IND-CPA game.

Definition 2.11. Given a generator $\alpha \in \mathbb{G}$, a cyclic group of order q, and two group elements α^a and α^b, the Decisional Diffie-Hellman Problem (DDH) is the problem of distinguishing between tuples of the form $(\alpha^a, \alpha^b, \alpha^{ab})$ and $(\alpha^a, \alpha^b, \alpha^c)$ for some random integer c, without knowledge of a and b.

As with CDH, a solution to DLP can be polynomially reduced to a solution to DDH; however, unlike CDH, there exist groups for which DDH is tractable, but DLP is not. One such example is an elliptic curve group that supports a bilinear pairing (described in the next

section). Given a pairing e and elements α^a, α^b, and α^c, then one can compute $e(\alpha^a, \alpha^b) = e(\alpha, \alpha)^{ab}$ and $e(\alpha, \alpha^c) = e(\alpha, \alpha)^c$. If the two values are equal then $ab = c$, otherwise $ab \neq c$.

Assuming that groups are chosen for which the index-calculus method does not work, each of the discrete logarithm problems can be parameterized by ϵ, the bitlength of the size of the group. Thus, recalling the running time of DLP algorithms, ϵ must be chosen so that solving a problem of order $O(\sqrt{2^\epsilon})$ is intractable.

2.1.6 Bilinear Groups and Elliptic Curves

Many mathematical functions are linear, that is, the relation $f(ax) = af(x)$ holds for any x in the domain of f, and for any constant a. The concept of a linear function can be extended to higher dimensions, with the two-dimensional, or bilinear case, being of particular interest.

Definition 2.12. Let $\mathbb{G}$ and $\mathbb{G}_1$ be two cyclic groups of order n with g a generator of $\mathbb{G}$. A map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_1$ is said to be bilinear if $e(g, g)$ is a generator of $\mathbb{G}_1$ and

$$e(u^a, v^b) = e(u, v)^{ab}$$

for all $u, v \in \mathbb{G}$ and all $a, b \in \mathbb{Z}$.

In cryptography, bilinear functions, are usually generated from elliptic curve groups.

Definition 2.13. An elliptic curve over the a field of characteristic not 2 or 3 is a plane algebraic curve defined by an equation of the form

$$y^2 = x^3 + ax + b$$

The set of points on an elliptic curve over a field whose characteristic is not 2 or 3, together with a special point an infinity, forms an abelian group. Boneh, Goh, and Nissim [Boneh et al., 2005] describe the following method for constructing a bilinear pairing over such a group of order n, where n is square free and not divisible by 3:

1. Find the smallest integer l such that $p = ln - 1$ is prime and $p = 2 \bmod 3$.
2. The curve $y^2 = x^3 + 1$ defined over $\mathbb{F}_p$ has $p + 1 = ln$ points in $\mathbb{F}_p$, and thus has a subgroup of order n, denoted by $\mathbb{G}_p$.
3. If $\mathbb{G}_1$ is the subgroup of order n of $\mathbb{F}_p^2$, then the modified Weil pairing on the curve gives a bilinear map from $\mathbb{G} \times \mathbb{G}$ to $\mathbb{G}_1$.

The Weil pairing is a method of constructing a bilinear pairing on a special subgroup of an elliptic curve group over a field.

When presenting an elliptic curve Paillier cryptosystem, Galbraith [Galbraith, 2002] uses a more general representation of elliptic curves, as the set of points $(x : y : z)$ such that $x, y, z \in \mathbb{R}$, a commutative ring with unity, and

$$y^2z = x^3axz^2 + bz^3$$

where $6(4a^3 + 27b^2) \in \mathbb{R}$ is invertible.

2.2 Homomorphic Encryption With The Ability to Compute on Encrypted Data

Homomorphic cryptography has long been the focus of many researchers because this technique allows computing operations to be performed on encrypted data. Using this technique, anyone can let an untrusted party perform a specific algebraic operation on their plaintext by applying the same or different operation on the ciphertext without revealing the plaintext. This characteristic of homomorphic encryption has a great potential for practical applications because any computer algorithm can be converted to a series of arithmetic operations. Therefore, if a homomorphic encryption scheme can allow an arbitrary number of additions and multiplications on the ciphertext, it would be possible to execute many computing application securely on the cloud.

Homomorphic encryption can be better understood by studying the algebraic meaning of the terminology. The term *homomorphic* has its origin from a mathematical concept of abstract algebra called *homomorphism* [Gallian, 2009], which refers to a mapping φ between two groups $(G, \diamond)$ and $(H, *)$ such that

$$\varphi(x \diamond y) = \varphi(x) * \varphi(y)$$

for $x, y \in G$ and $\varphi(x), \varphi(y) \in H$. This concept can also be defined for similar algebraic objects such as rings with multiple operations. In the homomorphic cryptography context, we will define a mapping called **Encrypt** such that

$$\textbf{Encrypt}\colon \mathcal{P} \longrightarrow \mathcal{C}$$

in which $(\mathcal{P}, +, \cdot)$ is the ring of plaintexts and $(\mathcal{C}, \oplus, \otimes)$ is the ring of ciphertexts. The mapping **Encrypt** can be implemented as a probabilistic function that takes inputs from the plaintext

space $\mathcal{P}$ and produces outputs belonging to the ciphertext space $\mathcal{C}$ according to some probability density function. A space of randomised inputs $\mathcal{K}$ will the determine which of the many possible ciphertexts a plaintext may be mapped to. Therefore, we can broaden the definition of the encryption function to take into account the randomised inputs

$$\textbf{Encrypt}\colon \mathcal{P} \times \mathcal{K} \longrightarrow \mathcal{C}$$

A decryption function can be formally defined as a mapping called **Decrypt** from the ciphertext space $\mathcal{C}$ to the plaintext space $\mathcal{P}$. This is a deterministic, many-to-one function and is an inverse of the encryption function

$$\textbf{Decrypt} = \textbf{Encrypt}^{-1} : \mathcal{C} \longrightarrow \mathcal{P}$$

The definitions above allow us to formally describe a homomorphic encryption scheme with respect to an operation $\diamond$ on two elements m_1 and m_2 on the plaintext space $\mathcal{P}$ for some operation $*$ on the ciphertext space $\mathcal{C}$

$$\textbf{Decrypt}(\textbf{Encrypt}(m_1) * \textbf{Encrypt}(m_2)) = \textbf{Decrypt}(\textbf{Encrypt}(m_1 \diamond m_2)) = m_1 \diamond m_2$$

The significant potential of homomorphic encryption systems and the difficulty of constructing such schemes were first recognised by Rivest, Adleman, and Dertouzos in 1978 [Rivest et al., 1978a], within a year of the development of RSA. Over the years, many homomorphic encryption schemes were proposed that have the capability to compute on encrypted data. Many of those schemes, however, only allow a limited number of either addition or multiplication to be performed on ciphertexts. There are two types of homomorphic encryption techniques, partially and fully homomorphic schemes, which will be described in the following sections.

2.2.1 Partially homomorphic cryptosystems

An encryption scheme is considered to be partially homomorphic if its ciphertexts can be either added or multiplied an unlimited number of times but not both operations at the same time. There are two types of partially homomorphic techniques, namely, additively and multiplicatively homomorphic encryption.

Given two plaintext integers m_1 and m_2, an additively homomorphic encryption scheme has a binary operation $\oplus$ that guarantees the following condition is true for an addition of m_1 and m_2:

$$\textbf{Decrypt}(\textbf{Encrypt}(m_1) \oplus \textbf{Encrypt}(m_2)) = \textbf{Decrypt}(\textbf{Encrypt}(m_1 + m_2)) = m_1 + m_2$$

One example of additively homomorphic encryption is the additive variant of the ElGamal[ElGamal, 1985] public key encryption scheme. This technique requires a party named Bob to choose a random group element β and send it to another party named Alice along with the group order p and the generator α. Bob will keep the secret parameter n, which is used to calculate β by the equation $\beta = \alpha^n$. The public parameters help Alice to compute a temporary key $x = \alpha^k \bmod p$ and the masking key $\beta^k \bmod p$, which will be used to encrypt the plaintext message m according to the formula $y = \alpha^m \cdot \beta^k \bmod p$. The ciphertext that Bob will receive can be written as $c = (x, y)$ which has two components: an encrypted version of the plaintext message m and the ephemeral key x. To decrypt a message, Bob computes $x^{-n} \cdot y = \alpha^m \bmod p$ and then executes a brute force search to recover the message m, assuming that the task of retrieving m is efficiently computable. The decryption process is summarised in the following equation:

$$x^{-n} \cdot y \bmod p = (\alpha^k)^{-n} \cdot (\alpha^m \cdot \beta^k) = (\alpha^k)^{-n} \cdot (\alpha^m \cdot \alpha^{nk}) = \alpha^{-ak+m+ak} = a^m \bmod p$$

Using ElGamal scheme, an encryption of the sum of the plaintexts can be obtained by computing the products of each pair of elements in the ciphertexts. Given to plaintexts m_1 and m_2 and two corresponding ciphertexts $c_1 = \textbf{Encrypt}(m_1) = (x_1, y_1)$ and $c_2 = \textbf{Encrypt}(m_2) = (x_2, y_2)$, this additively homomorphic property can be described as follows:

$$\begin{aligned}(x_1 \cdot x_2, y_1 \cdot y_2) &= (\alpha^{k_1} \cdot \alpha^{k_2} \bmod p, \alpha^{m_1} \cdot \beta^{k_1} \cdot \alpha^{m_2} \cdot \beta^{k_2} \bmod p) \\ &= (\alpha^{k_1+k_2} \bmod p, \alpha^{m_1+m_2} \cdot \beta^{k_1+k_2} \bmod p) \\ &= \textbf{Encrypt}(m_1 + m_2)\end{aligned}$$

Other additively homomorphic scheme have also been proposed such as the Goldwasser-Micali [Goldwasser and Micali, 1982], Benaloh [Benaloh, 1994], Paillier [Paillier, 1999] schemes and the Boneh-Goh-Nissim scheme [Boneh et al., 2005], which allows an unlimited number of addition on ciphertexts, but only one multiplication. Additively homomorphic encryption can be applied, for example, in a simple electronic voting scheme [Micciancio, 2010], in which each vote m_i is assumed to be either 0 or 1 and is stored on the cloud in an encrypted form $c_i = \text{Encrypt}_k(m_i)$ using the homomorphic key k. The cloud will be responsible for the entire vote collection and tally process without requesting any decryption task from the users. When the vote collection period is over, the cloud will return the encrypted result $c = \text{Encrypt}_k(m_1 + ... + m_n)$ to be decrypted locally by a trusted authority possessing the decryption key to obtain the final result $m_1 + ... + m_n$

A multiplicative homomorphic cryptosystem can be defined in a similar way as the additively homomorphic scheme. In this case, there is a binary operation $\otimes$ that can be applied on the ciphertexts as if the corresponding plaintexts are multiplied. Given two plaintext m_1 and m_2, we have

$$\textbf{Decrypt}(\textbf{Encrypt}(m_1) \otimes \textbf{Encrypt}(m_2)) = \textbf{Decrypt}(\textbf{Encrypt}(m_1 \times m_2)) = m_1 \times m_2$$

A typical example of a multiplicative homomorphic encryption system is the unpadded RSA scheme [Rivest et al., 1978b], which was created in 1978 by Ron Rivest, Adi Shamir and Leonard Adleman. Initially, a party named Bob selects two large prime numbers p and q and computes $n = p \cdot q$. Next, the Euler's totient function $\phi(n)$ is computed to get the number of positive integers less than n and relatively prime to n. If n is a prime number, then $\phi(n) = n - 1$. Because Bob has chosen $n = p \cdot q$, we have $\phi(n) = \phi(p) \cdot \phi(q) = (p-1) \cdot (q-1)$. The public and private keys are selected by two integers a and b chosen by Bob such that $b = a^{-1} \mod \phi(n)$. The public key is equal to the smaller of the two numbers, $k_{pub} = a$ while the other becomes the private key $k_{pr} = b$. Bob will publish n and k_{pub} and keep p, q and k_{pr} as secrets. A party named Alice can encrypt a message m by computing $c = m^a \mod n$. Bob will then calculate $c^b \mod n$ to retrieve the plaintext message m, according to the following equation:

$$c^b \bmod n = (m^a)^b \bmod n = m^{a \cdot a^{-1}} \bmod n = m \bmod n$$

The multiplicative homomorphic property of this scheme can be seen when two plaintexts m_1 and m_2 are encrypted to get the corresponding ciphertexts $c_1 = \textbf{Encrypt}(m_1)$ and $c_2 = \textbf{Encrypt}(m_2)$. The product of c_1 and c_2 can be decrypted to get the product of m_1 and m_2 as shown below:

$$c_1 \cdot c_2 = m_1^a \cdot m_2^a \bmod n = (m_1 \cdot m_2)^a \bmod n = \textbf{Encrypt}(m_1 \cdot m_2)$$

2.3 A Survey of Homomorphic Cryptosystems

In this section, we describe with more details some well-known homomorphic cryptosystems. Their partially homomorphic characteristic has been widely applied in various secure applications. These schemes are also important building blocks for fully homomorphic cryptography schemes invented later. When presenting each cryptosystem in the following subsections, we will define the plaintext space, the ciphertext space, the key space and any required set of randomizers. We describe each cryptosystem in terms of the three algorithms, i.e. key generation

Gen, encryption *Enc* and decryption *Dec*. The key generation algorithm *Gen* will take a security parameter as input and create an instance of the cryptosystem. This security parameter will determine the difficulty of the underlying computational problem of the cryptosystem. The encryption *Enc* and decryption *Dec* algorithms will perform the encryption and decryption of messages given the public and private keys respectively.

2.3.1 The RSA Cryptosystem

The main complexity behind the RSA cryptosystem is the difficulty of calculating the e'th roots modulo n, where $n = pq$ for distinct odd primes p and q, when the factorization of n is unknown. Although factoring n is sufficient to break RSA, it is unknown if breaking RSA is equivalent to factoring n. Boneh and Venkatesan [Boneh and Venkatesan, 1998] have given evidence suggesting that the problems are not equivalent in general. More specifically, they show that it cannot be proved that solving the RSA problem is as hard as factoring using a straight line program when the public exponent is small, unless factoring is easy. Later, Brown [Brown, 2005] demonstrated a result in the opposite direction; namely, it was shown that there is no efficient algorithm that takes a small public RSA exponent and outputs a straight line program that solves the RSA Problem, unless factoring is easy.

Homomorphic Properties

The RSA cryptosystem provides the basic homomorphic operation of multiplication modulo n. Given two ciphertexts $c_1 = m_1^e \bmod n$ and $c_2 = m_2^e \bmod n$, then

$$\begin{aligned} c_1 c_2 \bmod n &= m_1^e m_2^e \bmod n \\ &= (m_1 m_2)^e \bmod n \end{aligned}$$

is an encryption of $m_1 m_2$. Furthermore, the RSA cryptosystem is not IND-CPA secure unless messages are randomly padded. If m_1 and m_2 are randomly padded as m_1' and m_2', then recovering $m_1 m_2$ from m_1' and m_2' is impossible. For this reason, the RSA cryptosystem is not often used for its homomorphic properties.

2.3.2 The ElGamal Cryptosystem

The ElGamal [ElGamal, 1984] cryptosystem is a public-key cryptosystem based on the problem of solving discrete logarithms. Let $g \in \mathbb{G}$ be a generator of a cyclic group of order q such that the Computational Diffie-Hellman Problem (CDH) and Decisional Diffie-Hellman Problem (DDH)

problems are hard in $\mathbb{G}$. Let $P = \mathbb{G}$, $C = \mathbb{G} \times \mathbb{G}$, $R = \mathbb{Z}_q$, and $K = \{(q, g, x, h) : h \equiv g^x \bmod q\}$, where q, g are as above. The key generation *Gen*, encryption *Enc* and decryption *Dec* functions are described as followed:

- Key generation function *Gen*: Given security parameter ϵ, *Gen(ϵ)* returns a generator g of a cyclic group $\mathbb{G}$ of order q, where q has bit-length ϵ, as well as $h = g^x$ where x is a random integer in $\mathbb{Z}_q$. The tuple $(\mathbb{G}, q, g, h)$ is the public key, and the tuple $(\mathbb{G}, q, g, x)$ is the private key.

- Encryption function *Enc*: Given a public key $pk = (\mathbb{G}, q, g, h)$ and a message $m \in P$, *Enc(pk, m)* chooses a random $y \in R$ and returns the ciphertext (c_1, c_2) where

$$c_1 = g^y \bmod q \text{ and } c_2 = m \cdot h^y \bmod q$$

- Decryption function *Dec*: Given a private key $pk = (\mathbb{G}, q, g, x)$ and a ciphertext $c = (c_1, c_2)$, $Dec(sk, c)$ returns the message:

$$\frac{c_2}{c_1^x} \bmod q = \frac{m \cdot h^y}{g^{xy}} \bmod q = \frac{m \cdot g^{xy}}{g^{xy}} \bmod q = m$$

Homomorphic Properties

The ElGamal cryptosystem provides the homomorphic operation of multiplication of two encrypted messages, as well as multiplication by a known constant and exponentiation by a known constant. Given ciphertexts (c_1, c_2) and (d_1, d_2) that are encryptions of m_1 and m_2, using random values y_1 and y_2, respectively, then:

$$(c_1 d_1, c_2 d_2) = (g^{y1} g^{y2}, (m_1 \cdot h^{y_1})(m_2 \cdot h^{y_2})) = (g^{y_1+y_2}, m_1 m_2 \cdot h^{y_1+y_2})$$

is a valid encryption of $m_1 m_2$. Furthermore, given a constant k, then

$$(c_1, k c_2) = (g^{y_1}, k m_1 \cdot h^{y_1})$$

is a valid encryption of $k m_1$, and

$$(c_1^k, c_2^k) = (g^{y_1 k}, m_1^k \cdot h^{y_1 k})$$

is a valid encryption of m_1^k. ElGamal ciphertexts can also be re-randomized without knowledge of the plaintext. Given a ciphertext (c_1, c_2) as before, choose a random integer $r \in \mathbb{Z}_q$ and calculate

$$(c_1 g^r, c_2 h^r) = (g^{y_1+r}, m \cdot h^{y_1+r})$$

which is a randomized valid encryption of m.

2.3.3 The Paillier Cryptosystem

The Paillier Cryptosystem [Paillier, 1999] is a cryptosystem which utilizes trapdoor discrete logarithms because decrypting a ciphertext in the Paillier cryptosystem requires the use of a logarithm function L. From group theory, the set of n'th residues of $\mathbb{Z}^*_{n^2}$ forms a subgroup of order $\phi(n)$, and each n'th residue has exactly n roots. The n'th residue classes of $\mathbb{Z}^*_{n^2}$ partition $\mathbb{Z}^*_{n^2}$ into $\phi(n)$ different classes, each of size n, and the class function $w \mapsto [w]_g$ maps an element of $\mathbb{Z}^*_{n^2}$ to one of these classes; that is, given $w = g^m y^n \in \mathbb{Z}_{n^2}$, the class function maps w to m. Because the computational problem of computing the class function, Class$[n]$, is thought to be hard, it is a natural building block for a cryptosystem. Additionally, the class function is homomorphic, meaning that any cryptosystem utilizing the class function as a decryption function will also be homomorphic. Let $n = pq$ for primes p and q, set $\lambda = \text{lcm}(p-1, q-1)$, and choose $g \in \mathbb{Z}^*_{n^2}$ such that $\gcd(L(g^\lambda \bmod n^2), n) = 1$, where $L(x) = \frac{x-1}{n}$. Then $P = R = \mathbb{Z}_n$, $C = \mathbb{Z}^*_{n^2}$, and $K = (n, g, p, q, \lambda)$. The key generation *Gen*, encryption *Enc* and decryption *Dec* functions are described as followed:

- Key generation function *Gen*: Given security parameter ϵ, $Gen(\epsilon)$ chooses two distinct $\frac{\varepsilon}{2}$-bit primes p and q, sets $n = pq$ and $\lambda = \text{lcm}(p-1, q-1)$, and chooses $g \in \mathbb{Z}^*_{n^2}$ such that $\gcd(L(g^\lambda \bmod n^2), n) = 1$. The tuple (n, g) is the public key, and the tuple (p, q, λ) is the private key.

- Encryption function *Enc*: Given a message $m \in P$ and a public key $pk = (n, g)$, $Enc(pk, m)$ chooses a random $r \in R$ and returns the ciphertext

$$c = g^m r^n \bmod n^2$$

- Decryption function *Dec*: Given a ciphertext $c \in C$ and a private key $sk = (p, q, \lambda)$, $Dec(sk, c)$ returns the message

$$m = \frac{L(c^\lambda \bmod n^2)}{L(g^\lambda \bmod n^2)} \bmod n \text{ where } L(x) = \frac{x-1}{n}$$

Homomorphic Properties

The Paillier cryptosystem supports homomorphic addition and subtraction of encrypted messages, as well as addition and subtraction of constants, and multiplication by a constant. Let $c_1 = g^{m_1} y_1^n \bmod n^2$ and $c_2 = g^{m_2} y_2^n \bmod n^2$. Then

$$c_1 c_2 \bmod n^2 = g^{m_1} y_1^n g^{m_2} y_2^n = g^{m_1+m_2} y_1 y_2^n \bmod n^2$$

is a valid encryption of $m_1 + m_2$,

$$c_1 g^k \bmod n^2 = g^{m_1} y_1^n g^k = g^{m_1+k} y_1^n \bmod n^2$$

is a valid encryption of $m_1 + k$, and

$$c_1^k \bmod n^2 = (g^{m_1} y_1^n)^k = g^{km_1} (y_1^k)^n \bmod n^2$$

is a valid encryption of km_1. Subtraction of encrypted messages and constants can be accomplished by computing $c_1 c_2^{-1} \bmod n^2$ and $c_1 g^{-k} \bmod n^2$ respectively. The Paillier cryptosystem also supports the re-randomization of messages such that if y' is a uniformly random element of R, then

$$c_1 y'^n \bmod n^2 = g^{m_1} y_1^n y'^n = g^{m_1} (y_1 y')^n \bmod n^2$$

is a uniformly random valid encryption of m_1.

2.3.4 Fully homomorphic cryptosystems

A cryptographic system which can support both addition and multiplication on ciphertexts is considered to be fully homomorphic. Such cryptosystems would be far more useful and powerful than partially homomorphic schemes. The idea of a fully homomorphic encryption scheme was first proposed by Rivest, Adleman, and Dertouzos in the late 1970s [Rivest et al., 1978a]. For more that thirty years, however, a solution to this problem remained unknown and it was unclear whether fully homomorphic encryption was even possible. During this period, the best result was the Boneh-Goh-Nissim cryptosystem [Boneh et al., 2005] which supports the secure evaluation of an unlimited number of addition operations but at most one multiplication on ciphertexts.

In 2009, IBM officially announced [IBM, 2009] the existence of the first credible solution to the problem of creating a fully homomorphic encryption scheme based on the research works [Gentry, 2009a, 2010] of one of the company's researchers, Craig Gentry. Initially, Gentry created a somewhat homomorphic encryption scheme using ideal-lattices [Micciancio and Regev, 2009] that allows only a limited number of addition and multiplication to be calculated on encrypted data. Each operation performed on the ciphertexts will create noise in the results. There is a noise threshold beyond which the resulting ciphertexts will be indecipherable. Gentry's idea was that, rather than relying on the structure of the encryption scheme to increase the threshold and perform more homomorphic operations, the resulting ciphertexts can be refreshed periodically to eliminate the noise. He then showed that by

modifying the somewhat homomorphic scheme slightly to make it *bootstrappable*, i.e. the scheme can actually evaluate its own decryption circuit, the noise can be reduced significantly so that more additions and multiplications can be performed without resulting in indecipherable outcomes.

The first fully homomorphic scheme proposed by Gentry, however, is highly unlikely to be practical. Users of his scheme are required to specify in advance the number of both operations they need to perform on ciphertexts. Because there is often a large number of operations that need to be calculated when converting a computer program into a Boolean circuit, both the size of the ciphertext and the complexity of the encryption and decryption operations will grow significantly. In his article [Gentry, 2009b], Gentry estimates that performing a fully homomorphic search with encrypted keywords would increase the amount of computing time by about a trillion compared to a typical search on plaintext data. In 2009, another fully homomorphic encryption scheme was published by Marten van Dijk et al. [Van Dijk et al., 2010], which is based on Gentry's approach but does not require ideal lattice. Instead, the authors presented a very simple somewhat homomorphic scheme that uses integers which can also be used as a building block of a fully homomorphic scheme. Therefore, this scheme is conceptually simpler than using Gentry's ideal lattice while it is more efficient and still preserves similar properties with regards to homomorphic operations. In 2010, Smart and Vercauteren presented another fully homomorphic scheme [Smart and Vercauteren, 2010]. Starting from a somewhat homomorphic scheme, the authors showed that their scheme has a smaller message expansion and key size, allowing efficient fully homomorphic encryption over any field of characteristic two.

2.4 Requirements of privacy-preserving cloud-based computing applications

In this section, we will first describe the basic characteristics of secure cloud-based applications. Then we will highlight the major security threats and requirements about how to secure those cloud-based applications. Finally, we will provide an overview of some solutions that have been introduced in the literature to protect the security of those cloud-based applications.

2.4.1 Overview of a secure privacy-preserving cloud-based applications

A cloud-based application will need the involvement of the following parties: the cloud service provider (CSP), the data users and the data owners. The CSP is responsible for providing

essential cloud-based services such as storage and computing infrastructure as well as other processing services such as machine learning or data mining capabilities. Data owners will upload their data to the cloud and use its storage service. The owners also want to use the cloud for advanced data processing capabilities data search, data mining and sharing. Because owners are using those cloud-based services, they can also be regarded as cloud users. The true cloud users can be defined as any party that consumes the cloud-based data and services. They can do operations such as downloading specific data from the cloud, requesting the cloud to perform a range from basic to advanced computing operations as well as accessing the shared data.

For cloud-based applications to be secure, data owners will often encrypt their data before storing on the cloud. Therefore, a trusted third party is often introduced in many secure cloud-based data storage and processing models. [Hur and Noh, 2011, Li et al., 2013]. This trusted party is responsible for generating cryptographic keys as well as updating or revoking any access key of users. From a perspective of the user, this third party can also be trusted for other tasks such as data verification, i.e. checking the data stored on the cloud on behalf of users.

2.4.2 Security and Privacy Threats to Cloud-based Applications

A cloud-based storage and processing application can be vulnerable to various types of threats from traditional security threats such as denial-of-service attacks, illegal intrusion or network eavesdropping to more specific cloud computing threats such as abuse of cloud services, side-channel attacks and virtualization vulnerabilities. We want to focus on user privacy and data security with the assumption that the cloud is an untrusted environment. We will describe with details the major sources of threats to user privacy, data contents and data integrity in cloud-based applications.

Typically, from the perspective of users, there are two types of adversaries that can pose security threats to cloud-based applications. One is from the insiders of the cloud service providers or participants of the applications, the other are external attackers or hackers completely outside the application environment. The threats from these adversaries may raise various security issues, such as data leakage or disclosure, unauthorized access, data corruption or loss, and user privacy breach [Xiao and Xiao, 2013].

Threats from insiders. In cloud-based applications, there are internal participants working in the cloud service provider or the trusted party. They can both the regarded as honest

but curious which is a popular cloud threat model in most existing schemes [Samarati, 2014, Li et al., 2014, Sun et al., 2013]. Honest but curious model means that while a cloud service provider can honestly follow the system protocols instructions set by the data owners as well as reliably and faithfully provide storage and computing services, but it is also curious about the content of the outsourced data and user access. The CSP has the capability and interest to learn about whatever information they can get from the data owners and cloud users, from the data stored on the cloud to the types of operations that are often executed by the participants. The personal information of users such as identity, preferences and habits are also an important part that the CSP is very interested in because this kind of information can help the CSP to provide more relevant information to their customers, increasing the effectiveness of their operations. Therefore, a secure cloud-based application must provide its users with the tools to protect their privacy and data confidentiality.

Threats from external attackers and other factors. There have been many major security breaches involving external malicious attackers who use a variety of attack techniques such as network eavesdropping, vulnerability scanning, and malware attacking. These attackers or hackers want to illegally obtain access privileges to the data stored and processed on the cloud. They may also want to change or delete the outsourced data, corrupting the data and disrupting the normal services of cloud-based applications. These attackers can be eavesdroppers listening to the communication line to infer about user activities or intruders trying to study the content of the cloud-based data. There are other factors which make the data stored on the cloud more vulnerable. For example, cloud-based data can be corrupted or erased due to a hardware or software failures, software updates, configuration errors or bugs in the applications processing such data. Furthermore, the availability of cloud-based data might be affected by natural or man-made disasters like earthquakes, fires, and power failures[Bowers et al., 2011, Wang et al., 2013].

2.4.3 Essential Security Requirements of Cloud-based Applications

To address the security challenges and threats posed by various factors mentioned previously such as malicious insiders and attackers as well as to provide cloud users and data owners with strong protection of privacy, data confidentiality and integrity, any cloud-based application should be designed to meet the following security requirements:

- **Data Confidentiality**. It means that unauthorized users cannot access the content of the protected data. After the data is outsourced to the cloud, the data owners no longer

have a direct control as they used to have. Therefore, data confidentiality property will ensure that any user accessing the sensitive data will have to be authorized first to determine suitable access rights. Any unauthorized users, including the CSP will not be able to obtain any information about the data. However, data confidentiality should not affect the usability and availability of cloud-based services. Data owners and users should be able to fully utilize storage and processing capabilities of the cloud such as search, computation and data sharing without leaking data content to CSP or any unauthorized parties.

- **Privacy Preservation**. This is an important feature of any cloud-based application which will determine whether the user will trust and use that application. The expectation of user is that their identity information will be protected when using storage and computing services on the cloud. Users will not want their behaviours and habits to be inferred by any other internal or external parties. For example, when a user sends keywords to a cloud-based database to make queries over the outsourced data, both the keywords and the query results returned by the cloud should not be exposed to unauthorized parties.

- **Data Access Control**. The owners of data stored on the cloud should be able to control access to their data. This requirement can be achieved when cloud-based applications provide users with technical measures to specify access rights. Another advanced access control feature that should be provided to data owners is fine-grained access control. It means each user will be granted different set of data access rights depending on the access context and what kind of data the user want to access.

- **Data Integrity**. Ensuring data integrity is also an important requirement of data security. Any application which provides cloud-based data storage must guarantee that the integrity of data stored on their cloud servers will not be compromised in any way, for example, being tampered with, fabricated, or being deleted without proper authorisation. Data owners should be provided with essential tools to monitor the integrity of their data. Therefore, if there is any malicious operations which affect the integrity of data, the data owners will be able to detect immediately and take appropriate actions to fix and prevent further damages. Furthermore, data integrity protection operations should not hinder data accessibility. If a portion of data is corrupted or damaged, the rest of the data should still be accessible by the users of the cloud-based applications.

2.5 Review of Existing Secure Privacy-preserving Data Access and Computing Schemes on Cloud.

When a user or a business organisation use cloud storage and computing services, the cloud application service provider must preserve the privacy of their customers. User privacy includes all sensitive information about a user, a group of users or an organisation that must be kept secret from other unauthorized parties such as personal identifiable information, health information or information related to business activities. This privacy protection requirement exists in many components and operations of a cloud-based application, for example, in user behaviours, access pattern, keywords or sensitive outsourced data storage. There are many existing related research work about these aspects of privacy preservation that we want to cover in this section such as the privacy of user identity, the privacy of the queries that users send to cloud-based database and the privacy of user access patterns. Implementing solutions to protect each of those aspects of privacy requires various security techniques and models. When the access control of data is shared within a group, various group and ring signature techniques have been used to keep user identity information secret [Wang et al., 2012a,b]. Anonymous access techniques also play a significant role in protecting user identity in order to avoid identity leakage when data access control is enforced. Privacy of user queries can be protected by the use of secure indexes [De Capitani di Vimercati et al., 2011], virtual/dummy keywords [Sun et al., 2013], or random trapdoors [Cao et al., 2014]. Finally, the privacy protection of access pattern can be achieved by several approaches such as dynamically allocated data structure [Yang and Jia, 2012] and private information retrieval (PIR) [Chor et al., 1998]. In the following sections, we will go into the details of each category of cloud-based user privacy preservation.

2.5.1 Protecting User Identity on Cloud

On of the major privacy preservation requirements is how to protect a user identity when a user wants to authorize to access data and computing services on cloud. Users want to keep their personal identifiable information secret or just reveal as little information as possible to the cloud service provider or any trusted third party. To achieve this goal, there are several research works that have been proposed [Nabeel and Bertino, 2014, Nabeel et al., 2013, Jung et al., 2013], which can be divided into two categories, i.e providing anonymous access and group signature schemes.

Providing Anonymous Access. To enable identity protection and enforce data access control in untrusted servers, the use of oblivious attribute certificates (OACerts) is often propsed to implement a server access control policy. The attribute value in an OACert certification can be used to perform access control instead of a user identity so that the cloud service provider does not know about any attribute values. There is also an oblivious commitment-based envelope, which is a cryptographic protocol to deliver a message to users in an oblivious way. This approach is the basis for the research work of [Nabeel et al., 2011] to propose a cloud-based storage with fine-grained and flexible access control while being able to minimize the user information learnt by the cloud service provider. Another research work by [Nabeel and Bertino, 2014] also adopts the oblivious commitment-based envelope to enable user identity privacy protection. Access control is enforced using a token-based system. The privacy of user uploading data can also be protected by an approach proposed by [Shang et al., 2010]. The researchers use atribute-based cryptography together with oblivious commitment-based envelope to create fine-grained access control policies so that the data publisher will not learn anything about user identity attributes and the choice of access control policies. Atribute-based encryption is also used in another anonymous access control scheme proposed by [Jung et al., 2013]. The authors do not use a centralized attribute authority, instead they use multiple authorities while still maintaining tolerance to attacks on those authorities, effectively solving the problem of identity privacy preservation.

Group Signature Schemes. The privacy of user identity information can also be preserved by using group signature schemes. In their work [Wang et al., 2012a,b], the authors propose the use of group and ring signature techniques to prevent a third party from obtaining any user information during the signature verification process. Public key cryptography is an important part of this scheme. When there is a change in group membership, such as leaving or joining a group, the private key required to generate signatures has to be reproduced and delivered to each authorized member of the group while updating the corresponding public key. Therefore, one limitation of this research work is the scalability of the scheme especially when there is a large number of members in a group or if there are frequent changes in group membership. To address the privacy-preserving requirements of public verification for shared cloud data, another scheme proposed by [Wang et al., 2013] has shown how dynamic broadcast encryption can be used so that group private key can be securely distributed to existing users while outsourcing the recomputation of signatures on shared data to the cloud by exploiting proxy signatures. Therefore, privacy-preserving updates can be achieved with efficiency in large size groups.

2.5.2 Protecting User Behaviours and Access Patterns on Cloud

When users access data and use many computing services on cloud, their usage behaviours can form access patterns. If cloud-based applications do not have robust mechanisms to protect these access patterns, malicious third parties can obtain a great deal of information from these patterns such as user activities, browsing habits and frequency, access privileges, etc. Therefore, an important requirement is to protect cloud user behaviours and access patterns, preventing cloud service providers or unauthorized parties from gathering sensitive user information according to their access requests and corresponding server responses. There are several research approaches to address this requirements such as using dynamically allocated data structure and private information retrieval techniques, which will be described in the following paragraphs.

Dynamically Allocated Data Structure. The protection of access pattern can be implemented by using dynamic data allocation. The goal is to remove the correlation of physical blocks and the real contents. One research work [De Capitani di Vimercati et al., 2011] uses a shuffle index as a type of dynamic data allocation structure, providing pattern confidentiality by combining data shuffling, cover searches and cached searches. There are many advantages of their approach, i.e. data shuffling breaks the correlation of data blocks and tree nodes, while cover searches will hide the target of an access within a set of other potential targets and cached searches make repeated access to node content indistinguishable from non repeated access for a server. Another research work [di Vimercati et al., 2013] extends this scheme to enable concurrent accesses to data and searches over multiple indexes. Concurrency is achieved by using several distinguishing versions of the data index. These versions are coordinated and applied to the main data structure at regular intervals. Multiple indexes are also supported when the primary index is combined with multiple secondary shuffle indexes in a single data structure. Using such different versions and a combined index structure, the extended solution guarantees access pattern privacy.

Private Information Retrieval. Another important privacy preserving feature in cloud-based applications is how to protect user access requests and server responses. This feature is called private information retrieval (PIR) which is often divided into two types, i.e. computational and information theoretic PIR. To protect user privacy, the computational PIR approach relies on some mathematical problems that are already proven to be computationally hard. This approach can save the communication bandwidth for data retrieval because it does not require complete data copies. One research work [Yekhanin, 2010] has shown that this ap-

proach can be used to effectively prevent collusion attacks among multiple servers. However, it also introduces high computational overhead. Another method of PIR is the information theoretic approach which relies on the storage of multiple copies of the data stored in different distributed servers that cannot communicate with each other. Each server participating in this PIR protocol is guaranteed to get no information about what user access. The research work of [Shah et al., 2011] constructs locally decodable codes and converting the codes to PIRs as well as proposes an erasure-coded PIR that reduces replica storage and communication overhead.

2.5.3 Protecting the Privacy of User Queries on Cloud

There have been many research works which focus on query privacy preservation over ciphertexts outsourced to the cloud. The privacy of queries can be divided into three categories, i.e. trapdoor unlinkability, index privacy and keyword privacy. Trapdoor unlinkability ensures relationships of trapdoor cannot be inferred from multiple query trapdoors, implying that it is not decidable no matter whether different trapdoors of query requests are originated from the same keywords. Index privacy means that index information of original sensitive data is kept secret, while keyword privacy guarantees that any query keyword cannot be deduced from trapdoors of query requests. In the following paragraphs, we will provide an overview of these types of query privacy protection.

Trapdoor Unlinkability and Keywords Protection. To search on encrypted data, a common approach is using a trapdoor generation function to transform query keywords into special trapdoors before searching on encrypted data. There are several research schemes about trapdoor unlinkability and privacy protection, for example, some works [Sun et al., 2013, Örencik and Savaş, 2014] propose the use of virtual keywords and random numbers during keyword search, expanding query scopes, and randomizing query results. One research work by [Sun et al., 2013] has shown an effective way to protect query keywords and trapdoor by utilizing a variety of techniques such as adding virtual keywords in queries and using a separate query vector while introducing random factors to calculate the similarity of query results. Having a random number as input to the trapdoor function, the same query keywords will produce two different query trapdoors, ensuring the unlinkability between index privacy and query trapdoors. Another research by [Hu et al., 2014] has shown an oblivious index traversal framework and has integrated various techniques in the framework such as homomorphic encryption, conditional oblivious transfer and database indexing techniques to provide secure search on key-value stored. A secure search protocol is developed within this framework using

B+ tree and R tree indexes. One prominent feature of this protocol is that during query processing, it can resist traceablility from service providers and hence providing keyword privacy preservation.

Index Privacy Protection. To prevent the leakage of sensitive data during search, storage and other computational operations on cloud, it is very important to hide the relationships between original data and indexes. This requirement is often achieved by the use of a secure indexing function to map indexes to data. There are three main mapping modes, one-to-one, one-to-many and many-to-one mapping modes. The first mapping mode is one-to-one, which is a direct index approach in which each plaintext is mapped to a different index and each index corresponds to a different plaintext. The second mapping mode is one-to-many, which is a flattened index approach in which each plaintext is mapped into a set of index values, while each index is mapped to a unique plaintext. Finally, the many-to-one mapping, which is bucket index where different plaintexts are mapped to the same index value, but each plaintext is mapped to only one index. There are several research works focusing on these mapping modes such as [Ceselli et al., 2005], [Wang and Lakshmanan, 2006] and [di Vimercati et al., 2013]. The three index modes still faces are potential leakage problems by exploiting additional horizontal knowledge and vertical knowledge. The risk of private information disclosure increases with the accuracy of the indices. Therefore a combination of the bucket index and flattened index can be used to completely prevent information leakage raised by indices.

Chapter 3

Design and Implementation of a Secure Cloud-based Billing Model for Smart Meters as an Internet of Things Using Homomorphic Cryptography

Abstract

Smart grids introduce many outstanding security and privacy issues, especially when smart meters are connected to public networks, creating an Internet of Things in which customer usage data is frequently exchanged and processed in large volumes. In this research, we propose a cloud-based data storage and processing model with the ability to preserve user privacy and confidentiality of smart meter data in a smart grid. This goal is achieved by encrypting smart meter data before storage on the cloud using a homomorphic asymmetric key cryptosystem. By applying the homomorphic feature of the cryptographic technique, we propose methods to allow most of the computing works of calculating customer invoices based on total electricity consumption to be done directly on encrypted data by the cloud. One of the outstanding features in our model is the aggregation of encrypted smart meter readings using fixed-point number arithmetic. To test the feasibility of our model, we conducted many experiments to estimate the number of homomorphic additions to be performed by the cloud and the computation time in different billing periods using data from the Smart project, in which smart grid readings were continuously collected from

different households in every second within two months and electricity usage data collected every minute from 400 anonymous houses in one day. We also propose a parallel version of our billing algorithm to utilise the processing capability of multi-core processors in cloud servers so that computation time is reduced significantly compared to using our sequential algorithm. Our research works and experiments demonstrate clearly how cloud services can strengthen the security, privacy and efficiency of privacy-sensitive data frequently exchanged and processed in an Internet of Things where smart meters communicates directly with public networks.

3.1 Introduction

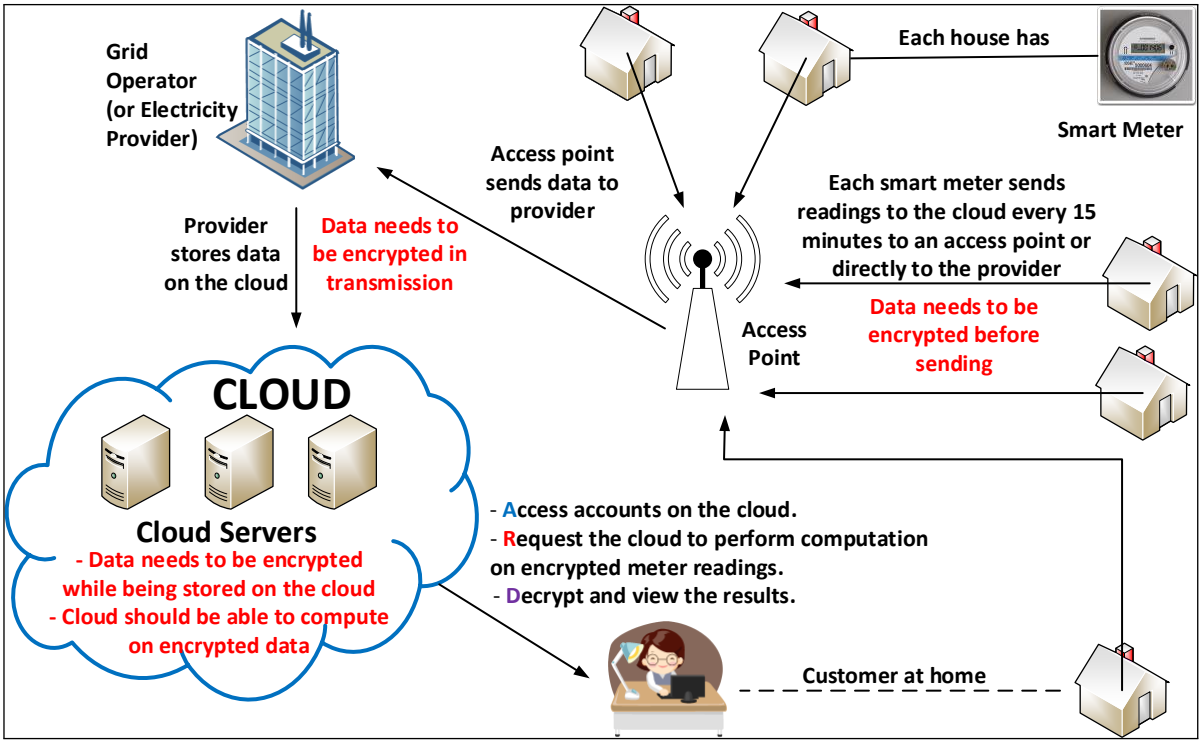
As already discussed in Chapter 1 of this thesis, we want to apply homomorphic cryptography to create secure and privacy-preserving protocols for cloud-based applications. Especially, in this chapter, we focus on the smart grid system. The question that we want to address in this chapter is how to protect user privacy at the lowest level when a smart-grid depends on a cloud-based system for data storage and processing. Today, the energy industry is facing new challenges in meeting the rising demands of electricity due to an increasing number of electrical devices. Furthermore, consumers often tend to use a large amount of electricity at the same time, for example, in the hottest summer or coldest winter days, creating peak periods when the electricity grid can be put under extreme stress that might cause blackout. One way to solve this problem is to keep building more generators and installing more poles and wires. However, this solution is expensive to implement given just a few such peak periods during a year and the fact that electricity bills are already going up and are predicted to keep rising due to other factors such as increasing service charges, the deregulation of the electricity market, or the monopoly in energy transmission and distribution.

To manage the supply and consumption of electricity more efficiently, innovations in digital communication, sensoring and metering technologies have been employed to complement the existing electricity grid, creating a more interactive, two-way grid system called the smart grid. On such grids, traditional electricity meters are replaced by modernised smart meters, which reside within the customer facility but are owned and managed directly by a grid operator via bidirectional communication links. Unlike older meter generations which only measure total electricity consumption in a month, a quarter, or year, smart meters can monitor and record electricity usage in much shorter time intervals, i.e. fifteen or thirty minutes, and automatically send this data to a grid operator. The report frequency and other parameters of a smart meter can be set remotely by the grid operator, making it possible to monitor almost in real-time the

consumption of electricity.

Smart grid technologies have brought many great benefits to both providers and consumers of electricity. Governments and energy suppliers are now able to balance electricity generation with consumption through a system of billing in which customers are charged by how much energy they have consumed at different times of day using dynamic and flexible tariffs. Smart meters provide more accurate, up-to-date and fine-grained meter readings, helping energy suppliers to adjust electricity generation and prices according to demands, thereby reducing blackouts and improving the reliability of the electricity grid. Various types of energy monitor devices connected directly to smart meters via wireless links help consumers to view their usage history, the amount of electricity they are using, and the current tariff. Customers now have the choice about how and when to reduce their energy consumption and take control of their electricity costs. Furthermore, smart grids also make it more efficient to manage the distributed power generation such as local solar and wind generators.

Figure 3.1: The vision of our research, a smart grid as an Internet of Things containing a large number of smart meters, access points, users and a grid operator connected to a public network with the strong support of various cloud storage and processing services working directly on homomorphically encrypted data.



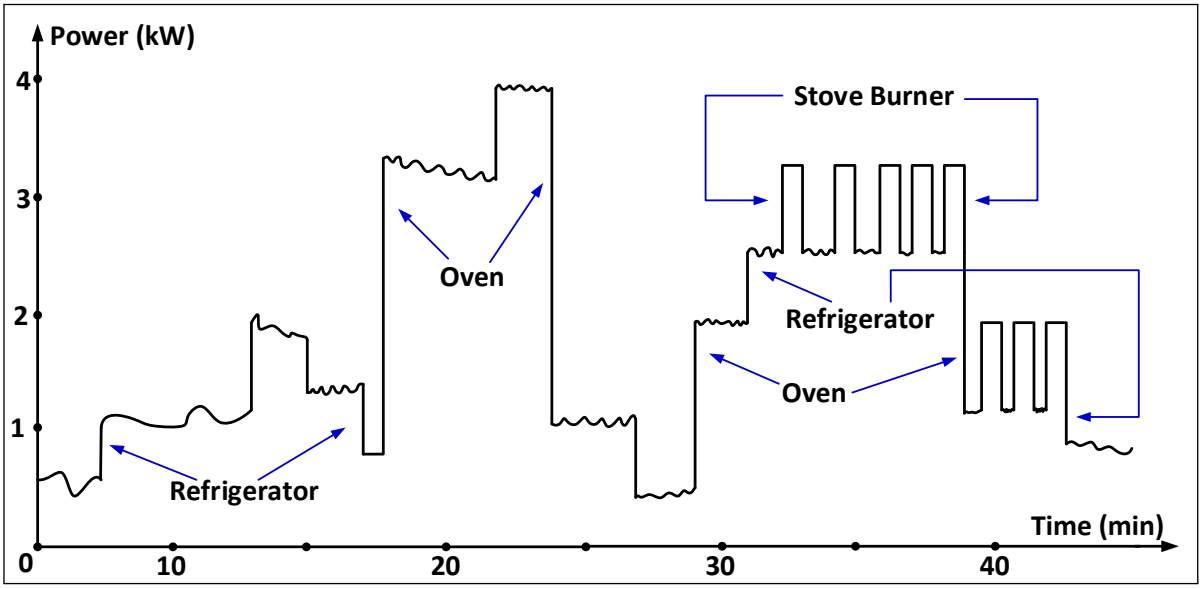
Despite the benefits to both the providers and consumers of electricity, a smart grid may also introduce many outstanding security and privacy issues, especially when smart meters are connected to public networks in which not only personally identifiable information but also energy consumption data relating to behaviours and movements of users is frequently exchanged and processed in large volumes. Figure 3.1 shows the vision of our research, in which we view a smart grid as an Internet of Things containing a large number of smart meters, access points, users, a grid operator and the cloud connected to a public network. The figure also shows how this Internet of Things can benefit from various cloud storage and processing services as long as security measures such as data encryption is implemented.

Furthermore, as the time interval of data collected by smart meters decreases to fifteen or thirty minutes, various load monitoring techniques[Zeifman and Roth, 2011, Weiss et al., 2012] can be employed to process unencrypted smart meter data to identify what electrical appliances, for example heaters, washing machines, refrigerators, air conditioners etc., are being used based on the electrical signature of those appliances[Ng et al., 2009, Akbar and Khan, 2007]. Figure 3.2 shows a power consumption trace of a customer [Quinn, 2009]. Many statistical tools and data mining techniques can be used to extract patterns from the fine-grained electricity consumption data to build user profiles and monitor user activities such as whether someone is at home, the habits of each family member, or what they do at particular moments, etc.[Fan et al., 2013, Acs and Castelluccia, 2011].

The security and privacy issues outlined above have prevented the cloud from reaching its full potential in supporting a smart grid, especially when many components of the grid are connected as an Internet of Things via the public network. Smart meters are constrained devices having limited capacity and incapable of performing complex computing tasks on energy usage data. Therefore, a smart grid needs a cloud computing system with its unlimited storage and processing capabilities to support complex computing tasks such as billings and analysing data. However, this goal cannot be accomplished fully without a suitable secure data storage and processing model providing the capability to compute on encrypted data. Therefore, we conducted the research described in this chapter to find out how the smart grid, as an Internet of Things can enjoy all the benefits of cloud computing while providing data owners with a strong guarantee of their privacy, the confidentiality of smart meter data and the reliability of the grid's information infrastructure.

Our Contributions. In this research, we propose a cloud-based data storage and processing model applicable for a smart grid with many components connected as an Internet of Things via the public network. Our model utilise the full strength of cloud computing and pre-

Figure 3.2: Tracking usage pattern using electricity consumption data [Quinn, 2009]



serves user privacy and the confidentiality of data exchanged on the grid. This goal is achieved by encrypting smart meter data before sending to the cloud for processing and storage using a homomorphic asymmetric key cryptosystem. Each smart meter is equipped with a set of private and public keys. The grid operator is given accessed to all the private keys while each household owner can only access the decryption key corresponding to their own smart meter. Using the homomorphic feature of the cryptographic technique, we create methods to enable most of the computing works to be done directly on encrypted data by the cloud, especially, we focus on the aggregation of encrypted smart meter readings directly on the cloud. During the homomorphic computation process, the cloud is allowed to access all public keys which are required by many homomorphic computing operations. However, the cloud does not have access to the private keys, hence, no decryption would be performed and no information would be leaked during the homomorphic computation process. The prominent features of our model are summarised as follows:

- User privacy and data confidentiality are protected by means of cryptography, especially when our model can store and process data mostly on the cloud. Only parties possessing the private keys can decrypt the data. An example of such parties are the grid operator because it needs to access the fine-grained meter readings to monitor the performance of the grid. A household owner can also decrypt their smart meter readings when they want

to know their total data usage and other statistics such as daily or hourly consumption. The electricity retailers, however, only need to know the total usage in a month or a quarter. Therefore, they are only given access to the data aggregation results and not the decryption keys.

- Our model allows the cloud to securely compute the aggregated energy consumption of a customer in a given period of time by performing homomorphic addition operation directly on an arbitrary number of encrypted fine-grained readings sent from the smart meter of that customer. The cloud does not need to perform any decryption during this computation process. Therefore, the privacy of the user and the confidentiality of the data are protected.

- Retailers are able to offer flexible pricing policies to consumers based on the time-varying costs of electricity procurement at the wholesale level. This feature is made possible because our model is designed so that the retailers can request the aggregated energy consumption of their customers during different time periods. The grid operator receives such requests from retailers, performs most of the data aggregation tasks directly on the cloud, then decrypts the results and sends them back to retailers.

- We propose a parallel version of our billing algorithm to utilise the processing capability of multi-core processors in cloud servers so that the computation time can be reduced significantly compared to using our sequential homomorphic addition algorithm. Using Amdahl's law and a metric called speedup, we measure how many times faster our parallel algorithm can actually run on some common processors with different number of cores and threads. We also demonstrate by many examples how an increase of speedup is determined by the size of the homomorphic expression as well as the number of threads used for the parallel computation process.

3.2 Background and Related Works

3.2.1 Fully Homomorphic Key Generation, Encryption and Decryption Algorithms

In our model, fully homomorphic encryption (FHE) technique is used as the main mechanism to encrypt data. Because the inputs are expressions containing signed integers, we need an efficient FHE scheme that can provide the most flexible ways to compute on the encrypted

version of these integers. Our goal can be achieved by converting the integer inputs to binary and then selecting an FHE technique to perform the encryption and decryption operations on each binary digit efficiently. Using this approach, we can apply existing logic circuits and binary manipulation techniques on the encrypted binaries. Therefore, we selected the FHE scheme proposed by Smart and Vercauteren (SV) because their scheme [Smart and Vercauteren, 2010] allows fully homomorphic encryption over any field of characteristic two and has small ciphertext size and key size.

At a high level, the FHE scheme used in this research requires a security parameter N specified by the user. The public key consists of a prime p and an integer $\alpha \bmod p$. The private key consists of an integer polynomial $Z(x)$ of degree $N-1$ to encrypt general binary polynomials of degree $N-1$. In our model, only binary digits are encrypted and hence, the private key consists of only an integer z of our choice. Generally, a message is encrypted by first encoding it as a binary polynomial and then a randomisation process occur by adding on two times a small random polynomial. Decryption is achieved when the resulting polynomial is evaluated at $\alpha \bmod p$. Therefore, the ciphertexts are integers modulo p regardless of whether bits or binary polynomials of degree $N-1$ have been encrypted. In our model, the plaintext message is a single bit. Therefore, decryption will require multiplying a ciphertext it by z and dividing the result by p. This rational number is then rounded to the nearest integer value, and subtract the result from the ciphertext. The plaintext is obtained by reducing this intermediate result modulo 2.

The Smart and Vercauteren [Smart and Vercauteren, 2010] homomorphic encryption scheme that we use to encrypt binary digits consists of five algorithm: **KeyGen**, **Encrypt**, **Decrypt**, **Add** and **Multiply**. This technique requires a set of input parameters (N, η, μ) in which $\eta = 2^{\sqrt{N}}$ and $\mu = \sqrt{N}$ and N is the security parameter specified by the user. Before each of the five algorithms can be described in details, two mathematical objects, i.e. a *polynomial ball* and a *polynomial half-ball*, are defined as follows:

$$\mathcal{B}_{2,N}(r) = \left\{ \sum_{i=0}^{N-1} a_i x^i : \sum_{i=0}^{N-1} a_i^2 \leq r^2 \right\}$$

$$B_{\infty,N}(r) = \left\{ \sum_{i=0}^{N-1} a_i x^i : -r \leq a_i \leq r \right\}$$

$$B^{+}_{\infty,N}(r) = \left\{ \sum_{i=0}^{N-1} a_i x^i : 0 \leq a_i \leq r \right\}$$

In the equations above, r is a postive integer, $\mathcal{B}_{2,N}(r)$ and $B_{\infty,N}(r)$ are polynomial balls while $B^{+}_{\infty,N}(r)$ is a polynomial half-ball. It can be seen that $B_{2,N}(r) \subset B_{\infty,N}(r)$ and $B_{\infty,N}(r) \subset B_{2,N}(\sqrt{N} \cdot r)$. In the following algorithms, the notation $a \longleftarrow b$ means assigning the value of b to a, while given a set A, writing $a \longleftarrow_R A$ means select a from the set A using a uniform distribution.

The **KeyGen** algorithm generates the public key **PK** and the secret key **SK**:

1. Set the plaintext space to be $\mathcal{P} = \{0, 1\}$
2. Choose a monic irreducible polynomial $F(x) \in Z[x]$ of degree N
3. Repeat

   ▷ $S(x) \longleftarrow_R \mathcal{B}_{\infty,N}(\eta/2)$
   ▷ $G(x) \longleftarrow 1 + 2 \cdot S(x)$
   ▷ $p \longleftarrow \text{resultant}(G(x), F(x))$
4. Until p is prime
5. $D(x) \longleftarrow \gcd(G(x), F(x))$ over $F_p[x]$
6. Let $\alpha \in F_p$ denote the unique root of $D(x)$
7. Apply the XGCD-algorithm over $Q[x]$ to obtain $Z(x) = \sum_{i=0}^{N-1} z_i x^i \in Z[x]$ such that

$$Z(x) \cdot G(x) = p \bmod F(x)$$

8. $B \longleftarrow z_0 \pmod{2p}$
9. The public key is $\mathbf{PK} = (p, \alpha)$, the private key is $\mathbf{SK} = (p, B)$

The **Encrypt** algorithm takes a plaintext message **M** and the public key **PK** as inputs to produce the ciphertext c:

1. Parse **PK** as (p, α)
2. If $M \notin 0, 1$ then abort
3. $R(x) \longleftarrow_R \mathcal{B}_{\infty,N}(\eta/2)$
4. $C(x) \longleftarrow M + 2 \cdot R(x)$

5. $c \longleftarrow C(\alpha) \bmod p$

6. Output c

The **Decrypt** algorithm decrypts the ciphertext **c** using the **PK** and returns the plaintext message **M**:

1. Parse **SK** as (p, B)

2. $M \longleftarrow (c - \lfloor c \cdot B/p \rceil) \bmod 2$

3. Output M

The homomorphic **Add** operation can be performed given two ciphertext $\mathbf{c_1}$, $\mathbf{c_2}$ and the public key **PK** as follows:

1. Parse **PK** as (p, α)

2. $c_3 \longleftarrow (c_1 + c_2) \bmod p$

3. Output c_3

Finally, the homomorphic multiplication operation is achieved by the performing the **Mult** algorithm that takes two ciphertext $\mathbf{c_1}$, $\mathbf{c_2}$ and the public key **PK** as inputs:

1. Parse **PK** as (p, α)

2. $c_3 \longleftarrow (c_1 \cdot c_2) \bmod p$

3. Output c_3

3.2.2 Encoding Integer Inputs Using The Two's Complement Numeric System

The inputs of our model are arithmetic expressions containing signed integers, which are converted to binary numbers and encrypted bit-by-bit using the SV fully homomorphic encryption scheme [Smart and Vercauteren, 2010]. We selected the two's complement format [Saha and Manna, 2009] to represent each integers so that we could apply the same circuits and computing techniques on the encrypted versions of both positive and negative numbers. At the binary level, the two's complement format allows us to treat positive and negative integers

Decimal	Two's Complement
127	0111 1111
64	0100 0000
1	0000 0001
0	0000 0000
-1	1111 1111
-64	1100 0000
-127	1000 0001
-128	1000 0000

Table 3.1: Some special values of an 8-bit two's complement numerical system

as unsigned integers when computing operations are performed. Only when we interpret the binary results, the sign bit of each integer will be taken into account.

The two's complement of an n-bit number is defined as the complement with respect to 2^n. Using this format, the most significant bit (MSB) is the sign bit, i.e. MSB = 0 if the integer is positive and MSB = 1 if the integer is negative. Therefore, an n-bit two's complement numeral system can represent every integer in the range -2^{n-1} to $2^{n-1} - 1$. Some special values of an 8-bit two's complement numerical system are shown in Table 3.1. As an example, let X = 109 be an integer that can be represented using $n = 8$ bits as the binary number 0110 1101. Denote $\overline{\mathrm{X}}$ the binary number obtained when the bits of X are inverted, i.e. 0s become 1s, and 1s become 0s, hence $\overline{\mathrm{X}}$ = 1001 0010. The two's complement of X will be $\overline{\mathrm{X}} + 1$ = 1001 0011, which is the unsigned representation of $2^n - \mathrm{X} = 2^8 - 109 = 147$.

Encoding the inputs using the two's-complement system gives our model many advantages, especially when encrypted bits are manipulated by homomorphic operations. One example of such advantages is that the number zero has only a single representation, obviating the subtleties associated with negative zero, which may exist in other encoding system such as ones'-complement. Most importantly, in a two's complement system, fundamental arithmetic operations of addition, subtraction, and multiplication are identical to those for unsigned binary numbers, making the system both simpler to implement and capable of easily handling higher precision arithmetic. For example, a two's complement of a positive number can be considered as the negative value of that number. Specifically, given two positive numbers X and Y, the expression X−Y can be calculated by adding X with the two's complement of Y. That is:

$$\mathrm{X} - \mathrm{Y} = \mathrm{X} + (\overline{\mathrm{Y}} + 1)$$

Fully Homomorphic Expressions with Ciphertexts	Corresponding Plaintext Expressions	Results
$\underbrace{c_0 + c_0 + c_0 + \ldots + c_0}_{n \text{ times}}$	$\underbrace{b_0 + b_0 + b_0 + \ldots + b_0}_{n \text{ times}}$	0
$\underbrace{c_1 \times c_1 \times c_1 \times \ldots \times c_1}_{n \text{ times}}$	$\underbrace{b_1 \times b_1 \times b_1 \times \ldots \times b_1}_{n \text{ times}}$	1
$c_1 \times c_0 + c_1 \times c_1 \times c_1$	$b_1 \times b_0 + b_1 \times b_1 \times b_1$	1

Table 3.2: Examples of fully homomorphic addition and multiplication operations, and the corresponding plaintext expressions

3.2.3 Encrypting Integer Inputs

After encoding all integers in the input arithmetic expression using the two's complement format, each of them can now be represented by an array of binary digits, ready for the bitwise fully homomorphic encryption process, as suggested in [Kaosar et al., 2012]. Specifically, an n-bit integer X $= (x_n, x_{n-1}, ..., x_1)$ where X $\in \mathbb{Z}$ and $x_i \in \{0, 1\}$, can be encrypted as shown in the following equation, using the encryption function **E** and the fully homomorphic public key **pk**:

$$\alpha = (\alpha_n, \alpha_{n-1}, ..., \alpha_1) = E_{pk}(X) = [E_{pk}(x_n), E_{pk}(x_{n-1}), ..., E_{pk}(x_1)]$$

Each ciphertext α_i is a large integer which represents an encrypted version of a binary digit. Similarly, the decryption function **D** can be used with the secret key **sk** to decrypt each ciphertext α_i and retrieve the binary array representing the integer X as follows:

$$X = (x_n, x_{n-1}, ..., x_1) = D_{sk}(\alpha) = [D_{sk}(\alpha_n), D_{sk}(\alpha_{n-1}), ..., [D_{sk}(\alpha_1)]$$

3.2.4 Fully Homomorphic Operations on Encrypted Binary Digits

At the core of our model, when two binary digit $b_0 = 0$ and $b_1 = 1$ are homomorphically encrypted as $c_0 = E_{pk}(b_0)$ and $c_1 = E_{pk}(b_1)$, the addition and multiplications of c_0 and c_1 can be performed an arbitrary number of times. The results of such computation, when decrypted, will be the same as performing similar computation on the plaintext b_0 and b_1. Some examples of such homomorphic computation are shown in Table 3.2.

The fully homomorphic addition and multiplication operations that can be applied on encrypted binary digits allow us to construct AND and XOR circuits with encrypted bits as inputs and outputs. From the truth tables of the XOR operation (Table 3.3), it can easily be seen that the addition of two binary digits without carry produces a result which is identical to the output obtained by applying these two bits as inputs to the XOR circuit. Similarly,

Input		Output
A	**B**	**A XOR B**
0	0	0
0	1	1
1	0	1
1	1	0

Table 3.3: XOR Truth Table

Input		Output
A	**B**	**A AND B**
0	0	0
0	1	0
1	0	0
1	1	1

Table 3.4: AND Truth Table

the truth table of the AND operation (Table 3.4) shows that passing two bits as inputs to the AND circuit will produce an output which is the same as multiplying these two bits together. With homomorphic encryption, we can add or multiply encrypted bits, making it possible to securely perform not only the XOR and AND operations, but also other circuits as long as they are based on the XOR and AND circuits.

3.2.5 Adding Encrypted Integers

Inputs			Outputs	
A	B	C_{in}	C_{out}	S
0	0	0	0	0
1	0	0	0	1
0	1	0	0	1
1	1	0	1	0
0	0	1	0	1
1	0	1	1	0
0	1	1	1	0
1	1	1	1	1

Table 3.5: Truth table of the full adder circuit

We can understand homomorphic addition of integers by first studying how they are added in plaintext. Before an addition operation of two integers happens, the operands are often converted to two's complement binaries and then the bits from each operands are added from the least significant to the most significant bit. A typical way to perform this binary addition is using a full adder circuit, which adds binary digits and accounts for values carried in and carried out. A diagram of a one-bit full adder circuit is shown in Figure 3.3, it's truth table is shown in Table 3.5, in which A and B are the operands and C_{in} is a bit carried in from the previous binary addition. The circuit produces a two-bit output, comprising of a carry out C_{out} and a sum S.

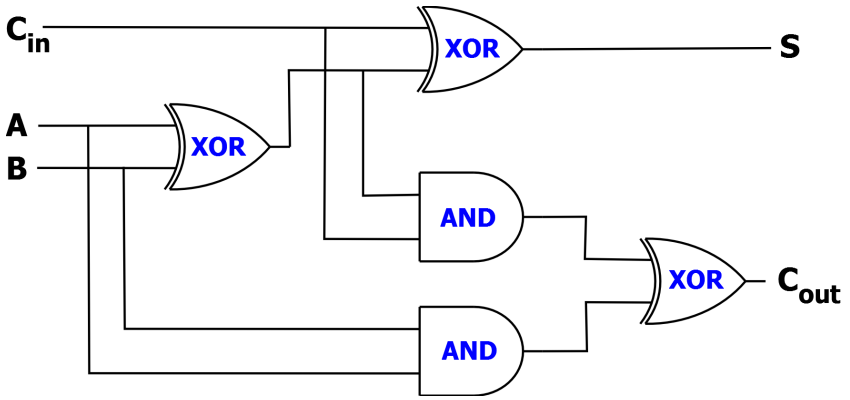


Figure 3.3: The full adder circuit

From the truth table of the full adder circuit, the sum S_i and the carry out bit C_i of an addition of two binary digits A_i and B_i can be described as follows (with $\oplus, \times$ and $+$ denote XOR, AND and OR respectively):

$$S_i = A_i \otimes B_i \otimes C_{\text{in}}$$
$$C_{\text{out}} = (C_{\text{in}} \times (A_i \otimes B_i)) + (A_i \times B_i) = (C_{\text{in}} \times (A_i \otimes B_i)) \otimes (A_i \times B_i)$$

These equations above show that the addition of two binary digits with a carry out can be performed with only XOR and AND circuit. In section 3.2.4, we have shown that the XOR and AND circuit can be evaluated securely with fully homomorphic encryption. Therefore, the full adder circuit can also take encrypted input bits and an encrypted carry in and produce an encrypted sum and carry out bit.

Adding two encrypted integers is equivalent to adding two arrays of encrypted binary digits with all carry bits taken into account. This can be achieved by using a ripple-carry adder [Burgess, 2011], which can combine multiple full adders to add n-bit numbers. Because a full adder inputs a carry in bit, which can be a carry out bit of a previous binary addition, many full adder can be chained together, so that the carry out bit from one full adder is the carry in bit of the next full adder, as shown in Figure 3.4. A ripple carry adder circuit can work with encrypted inputs and produce an encrypted result because its building blocks are full adders, which can be securely evaluated. Furthermore, the fully homomorphic nature of the underlying cryptographic scheme allows the encrypted carry out bit to be passed to other full adders an arbitrary number of times without affecting the accuracy of the final result.

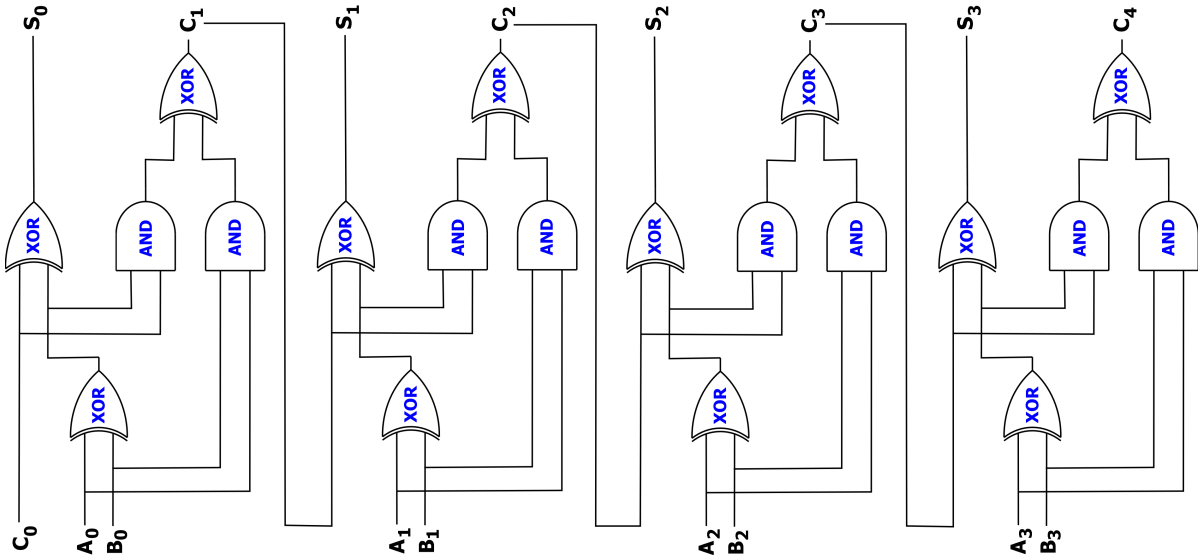


Figure 3.4: A 4-bit ripple carry adder

3.2.6 Related Works

There have been many research works focusing on the storage and processing of smart meter data while preserving user privacy and protecting data confidentiality. However, not many of those works provide detailed solutions and practical implementation about how the data is stored and processed in an encrypted form on the cloud.

Rial et al.[Rial and Danezis, 2011] propose a set of privacy-preserving protocols involving three parties: an electricity provider, a user agent and a simple tamper-evident meter. The data is encrypted by the smart meter using its symmetric key and stored in encrypted form in remote servers. To perform data aggregation and billing, the user will need to download the encrypted data and decrypt using a symmetric decryption key. A final bill will be sent to the provider with a zero-knowledge proof that ensures the calculation to be correct and leaks no additional information. However, this approach requires the user themselves or a third party that the user fully trusts to perform the computation. Cloud servers cannot compute on encrypted data. In a more realistic setting[Powercor], the electricity retailer would calculate the bills first and send to customers, given only the total energy consumption in fixed period of times.

In their work, Acs et al. [Acs and Castelluccia, 2011] describe a scheme allowing electricity providers to collect smart meter data periodically and derive aggregated statistics without learning anything about the activities of individual households. At each billing period, a smart

meter sends to its corresponding provider the readings that are mixed with random noise and encrypted to protect the privacy of a user. The provider can still compute the total electricity consumption of that household despite the amount of noise being added to the original data. However, this research work assumes that most of the computing tasks involved in finding the total energy consumption are performed by a provider rather than using the cloud, causing limitations in cases when the provider does not have enough computing power to aggregate fine-grained readings coming from a large number of meters or when to store and secure a huge amount of data over long periods of time.

Another research work has been proposed by Ruj et al. [Ruj and Nayak, 2013] in which a decentralized security framework was designed for smart grids with the capabilities to do both data aggregation and access control. The smart grid is divided into a hierarchy comprising a home, building and neighbouring area networks. Electricity usage data is collected and sent to substations along a path from lower to higher networks in the hierarchy under the monitor and control of remote terminal units. Security and user privacy is achieved by encrypting the data in the transmission process while aggregation tasks are performed on encrypted data due to the use of homomorphic encryption. While details of the access control component of the architecture are described very thoroughly, especially about how users can access the data through remote terminal units, the authors have not provided as many details about cloud-based data storage and how aggregation tasks are performed on encrypted data.

There are other research works providing different approaches to the security and privacy protection of smart meter data. In their work, Deng et al.[Deng and Yang, 2012] describe in detail how smart meters can register and authenticate their identities before a secure communication session can be set up with the data collector. These operations help to build a network of smart meters organised as an aggregation tree in which the data collector is the root node with information relating to the network structure and routing backbone. The author assume that smart meters are connected to one another and readings from one meter have to travel to other meters to reach the data collector. This feature is significantly different from our model in which each meter are independent and connected directly to the grid operator. In another research, Garcia et al.[Garcia and Jacobs, 2011] design protocols for basic communication with E-meters using elementary cryptographic techniques such as symmetric key encryption and digital signature. Their main goal is to learn the aggregated energy consumption of N consumers without revealing any information about the individual consumption of the users, even when the data collector is malicious. However, each meter M_i in their model must also know about the $N-1$ public keys of the other meters participating in the protocol. This is

a limitation because the smart meters in their model are programmed to abort the protocol when they cannot get enough meters with public keys willing to join a communication session.

In our research, we aim to complement existing literature by describing in details how data is stored on the cloud in encrypted formats. We also present a model showing how the homomorphic aggregation of encrypted smart meter readings is performed directly on the cloud. Furthermore, we implement the model and measure the computation time of our algorithm on real data sets.

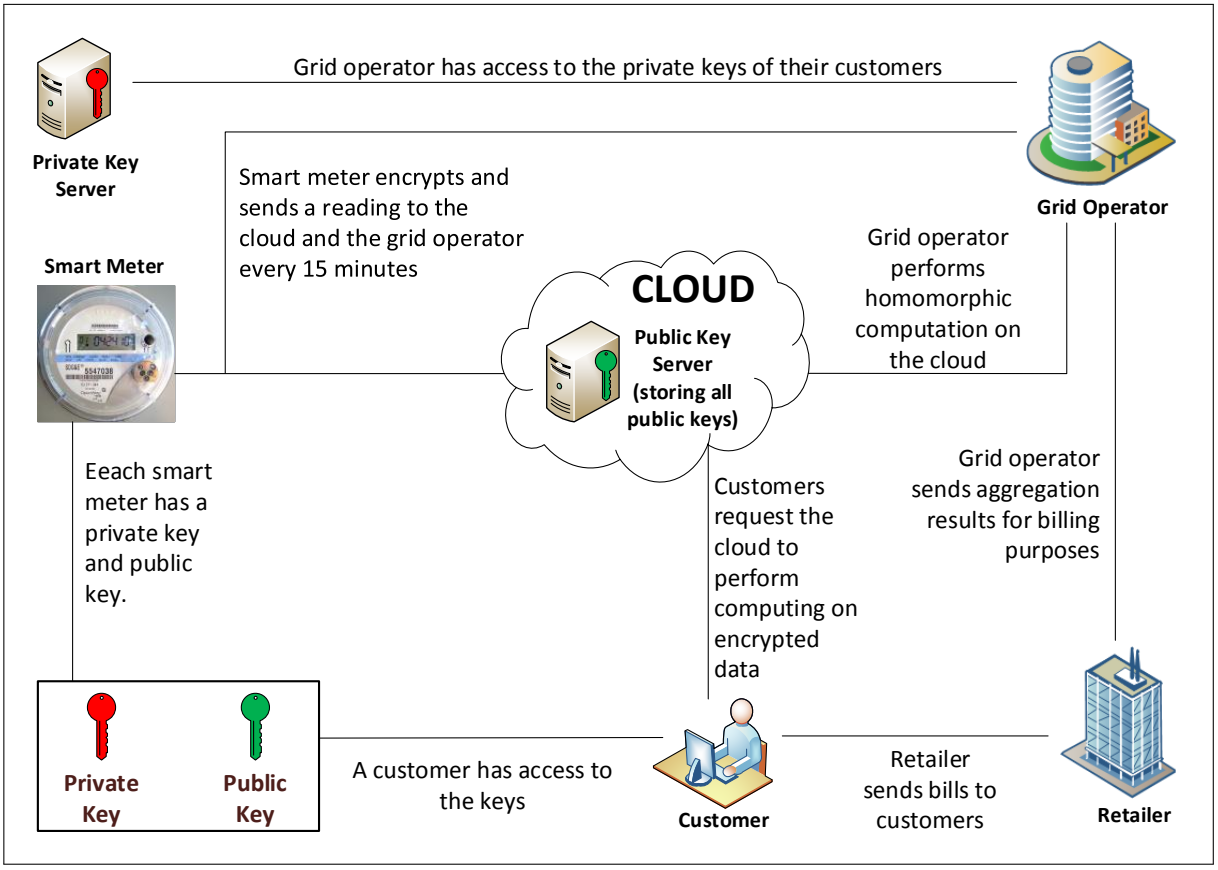
3.3 Cloud-based Smart Meter Data Storage And Processing

3.3.1 System Architecture

In our model, each smart meter is allocated an asymmetric key pair generated by a homomorphic cryptosystem. Every fifteen minutes, a meter reading is encrypted by the smart meter using the public key and sent to the grid operator for monitoring purposes and to the cloud for storage and homomorphic computation. The grid operator can access the private keys of all smart meters in the grid and decrypt fine-grained readings to monitor the grid performance. The cloud is responsible for performing most of the homomorphic computation over encrypted smart meter data by using the public keys of those meters. Retailers are responsible for issuing bills to their customers using the total amount of energy consumed in a month or a quarter. They are not allowed to access the private keys to decrypt fine-grained readings stored on the cloud. Instead, retailers send requests for data aggregation to the grid operator, which will use the cloud to perform the homomorphic aggregation tasks, decrypt the results and send them to retailers. Customers receive bills containing the electricity cost and other statistical results from their retailers. Our model allows customers to have access to the private keys of their smart meters. Therefore, customers can check their bills and perform other statistics on their usage data by asking the cloud to perform various homomorphic operations on their encrypted fine-grained readings and decrypting the results using their private keys. Figure 3.5 shows the details of our model.

The electricity consumption of each household is measured by a smart meter. Every fifteen minutes, the smart meter sends out a data package containing three types of data: a smart meter identification number, a timestamp and an encrypted reading. Each smart meter has an identification number granted by the grid operator when the smart meter is installed. This number is used to uniquely refer to the smart meter. Therefore, it must be included in each

Figure 3.5: The architecture of our model.

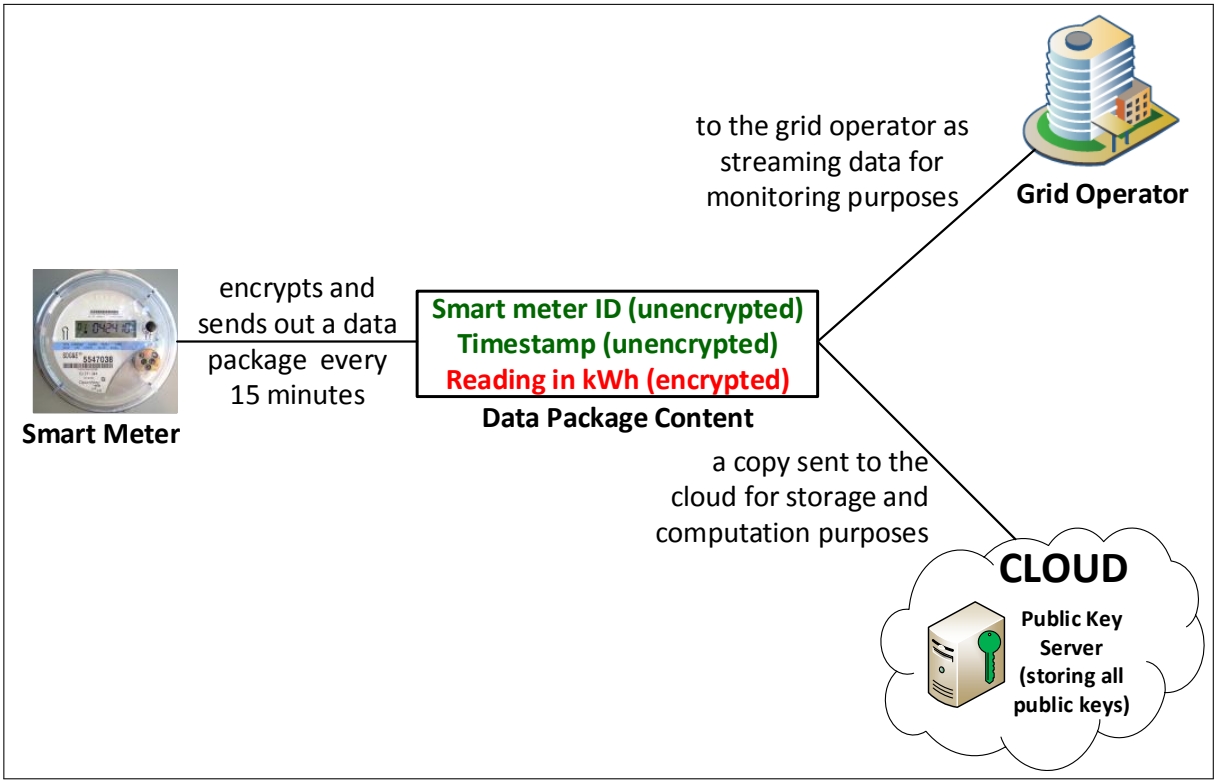


data package so that the grid operator can differentiate between data packages sent from different sources. The cloud also needs the identification number to identify smart meter data when processing queries. The timestamp indicates the moment when a reading is recorded. This value is used by the grid operator for performance monitoring purposes and for the cloud to know whether the reading was taken in the peak or off peak period. The identification number and timestamp are not encrypted. However, the smart meter reading taken every fifteen minutes is encrypted using the public key of the corresponding smart meter. Only the grid operator and the household owner of that smart meter have the private key to decrypt the readings. Encryption is required to secure the storage and processing of smart meter readings on the cloud. Figure 3.6 outlines the content of a data package sent out by a smart meter.

Supporting multiple pricing policies.

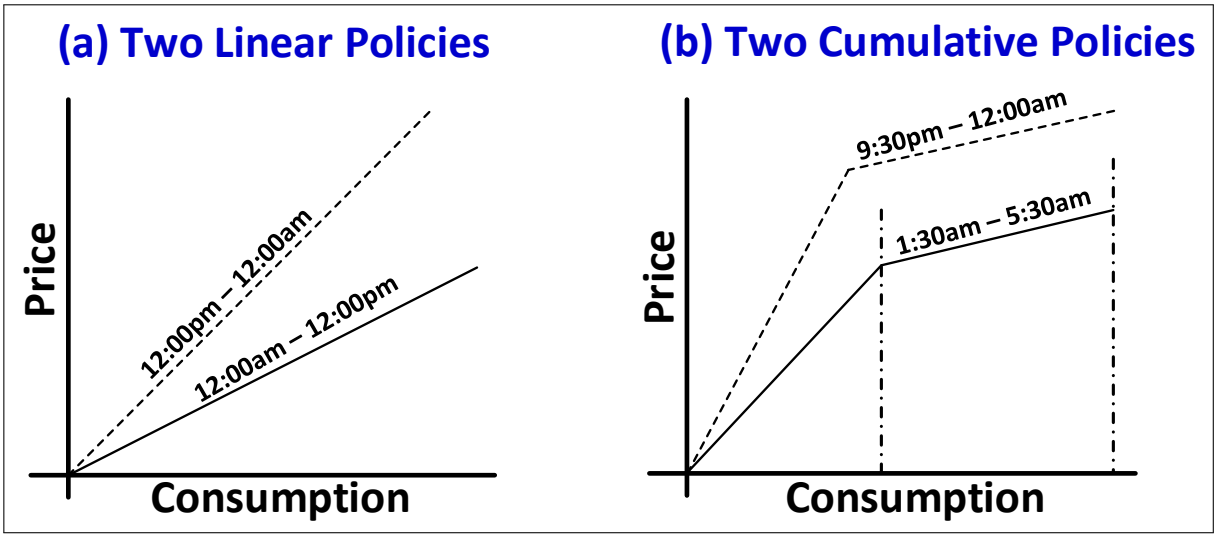
Our model allows a retailer to set flexible pricing policies based on different times of the day

Figure 3.6: Content of a data package sent by a smart meter every fifteen minutes



such as peak and off-peak period as shown in Figure 3.7, allowing customers to save more money and increasing the efficiency of electricity usage. A typical pricing policy $\mathcal{P}$ contains a smart meter reading x which shows the number of kWhs consumed during a fixed period of time (for example 15 minutes) and the time t where the reading was recorded. The time information t allows a retailer to decide a rate at which a customer will be charged for their usage x. Therefore, a pricing policy $\mathcal{P}$ can be regarded as a function $\mathcal{P} : (x, t) \rightarrow p$ that takes a reading x and a recording time t and outputs a price p. After each billing period, a total fee is computed by adding the prices corresponding to the total electricity consumption in that period, according to the formula: fee $= \sum_{i=1}^{n} p_i$, in which n is the total number of readings in a billing period and p_i is the price calculated for each reading. Our model is very similar to the practical smart grid architecture described in[Powercor] in that a retailer is not granted direct access to fine-grained smart meter readings of its customers. This design feature is reasonable because a retailer only requires the total energy consumption during a period for billing purposes while fine-grained readings are often required by the grid operator to monitor

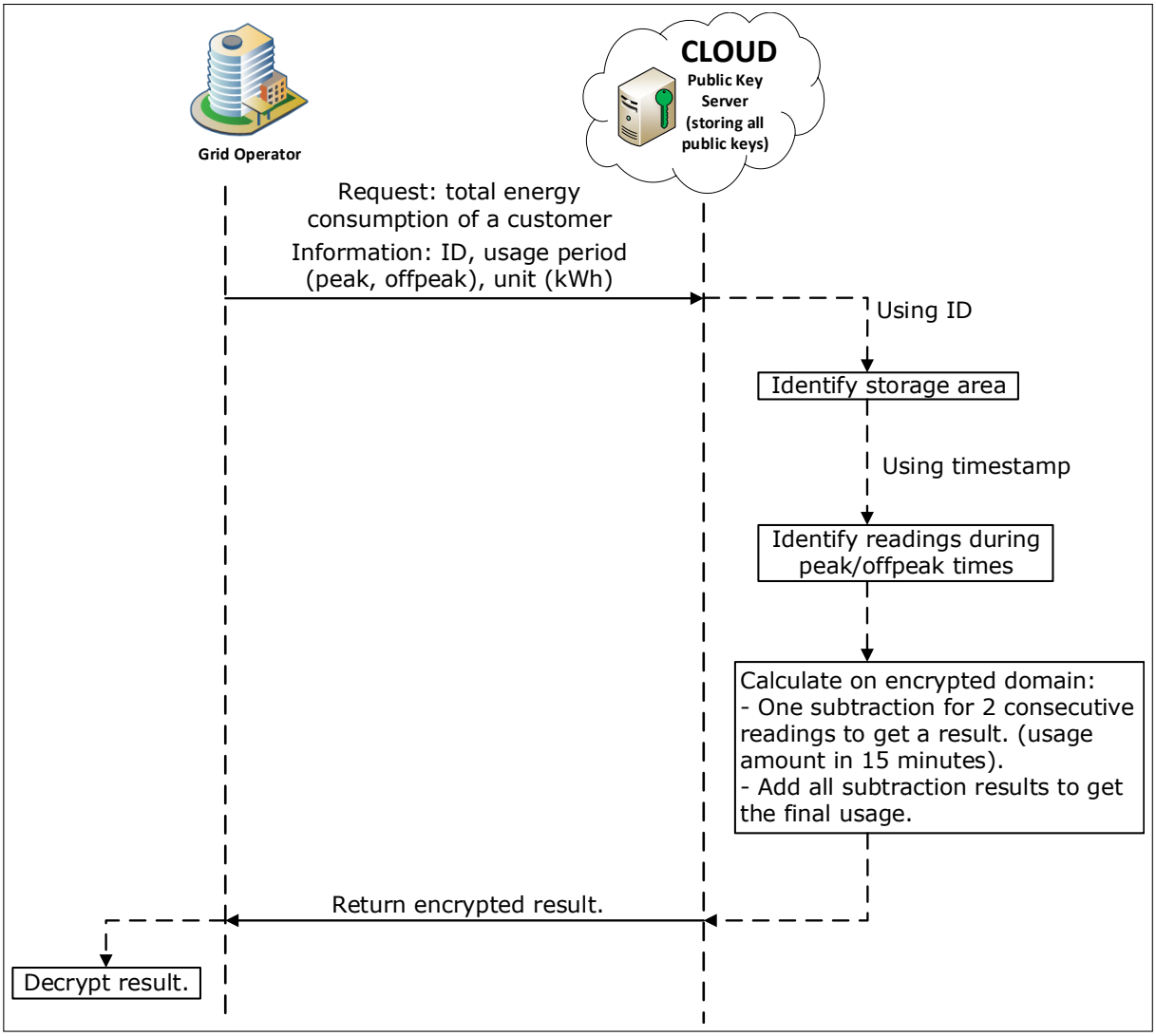
Figure 3.7: (a) A linear policy specifies the rate per unit consumption that is applied to determine the price to be paid for each measurement. (b) A cumulative policy specifies a rate per unit that is determined as a function of the consumption allowing non linear functions to be applied for pricing.



the grid performance. Furthermore, customers are far more likely to change their retailers rather than changing their grid operator because retailers constantly compete with each other to offer the best prices to customers while there are only a few grid operators which do not often deal with customers directly.

When a retailer wants to calculate a customer bill in a consumption period, it needs the total electricity figure of the customer during that period to multiply with the rates it offers. This total amount can be for a month or a quarter and can be divided into smaller sums corresponding to peak or off-peak periods as determined by various pricing policies set by the retailer. These total figures are calculated from fine-grained readings obtained from the customer's smart meter. Because the retailer cannot access those readings directly, it will send requests to the grid operator, which will instruct the cloud to perform most of the calculation tasks. This is where our model brings many benefits to customers, the grid operator and retailers in terms of data security, user privacy and efficiency. We find out a method to calculate the total figures directly on encrypted data, thereby allowing the cloud not only to store data securely but also to compute on encrypted data. Finally, the cloud outputs the total figures in encrypted forms and sends to the grid operator, which will decrypt and send to the retailer to complete the request.

Figure 3.8: Cloud-based calculation of total energy consumption on encrypted data for a customer



Calculating total energy consumption on the cloud using encrypted data.
After each billing period, retailers will need to calculate bills for each of their customers. They need the total energy consumption of each of the customer. However, retailers cannot access smart meter readings directly because they are encrypted and store on the cloud. According to our model, only the grid operator can access encrypted readings because it stores the private keys of all the smart meters. Therefore, retailers will send billing request to the grid operator which in turn will ask the cloud to perform most of the computation on encrypted data.

After being asked to calculate a total energy consumption for a customer in a period of time by a retailer, the grid operator will send a request to the cloud containing information to help the cloud to identify and calculate the data of that customer. The request will contain the identification details of the customer and the usage period during which the total energy consumption is calculated. Upon receiving the request, the cloud will use the customer identification number to locate the data related to that customer. Each smart meter reading is stored in encrypted form on the cloud and associated with a timestamp in plaintext. Based on the request usage period, the cloud decides which encrypted reading is relevant to the calculation. In the encrypted domain, the cloud will subtract two consecutive readings to get a result, for example, the total usage in 15 minutes, then all such results will be added to get the final usage in encrypted form. The cloud will send this encrypted result back to the grid operator which in turn will decrypt the result using the appropriate key and send the final unencrypted result to the retailer to apply appropriate rate to the total energy consumption. Figure 3.8 illustrates this process.

3.4 Experiments and Analysis

In our experiments, we used data from the Smart* project [Barker et al., 2012] in which smart grid data were continuously collected from three households in every second within two months. The authors also collected electricity usage data every minute from 400 anonymous houses for one day. This data together with some more data generated by us allowed various application scenarios to be simulated in which a smart meter in each house hold will send a total energy consumption in 15 minutes, allowing us to calculate the number of homomorphic addition operations and measure the time required by the cloud to calculate the total energy consumption in a given time period.

3.4.1 Data Structure and Key Generation

Before conducting experiments, we assumed that each smart meter reading is represented by a decimal number with two decimal places. Hence, we wanted all computing operations to be accurate to two decimal places. This was achieved by setting the resolution $\varepsilon = 0.001$. Hence, the number of bits QF used to represent the fractional part was calculated as:

$$\boxed{\text{QF} = \text{ceiling}\left(\log_2\left(\frac{1}{\varepsilon}\right)\right) = \text{ceiling}\left(\log_2\left(\frac{1}{0.001}\right)\right) = 10}$$

Using another 10 bits to present the integral part, we could compute with all decimal numbers from -512 to 511 up to two decimal places. We assumed that overflow would not occur by selecting numbers and calculating results within this range. This experimental setting allowed us to encode each smart meter reading as a 20-bit binary number, which was subsequently encrypted bit-by-bit using the Smart and Vercauteren [Smart and Vercauteren, 2010] homomorphic encryption scheme.

To test the feasibility of our model, we wrote two software modules using the C programming language to represent a client and the cloud. The client module was responsible for encryption and decryption tasks while homomorphic computing operations were performed by the cloud module. The cloud would use public keys to perform fully homomorphic operations and return encrypted results to clients. Each smart meter was identified on the cloud using a smart meter identification number. The readings were encrypted using homomorphic encryption while the ID and timestamp were stored in plaintext on the cloud. Table 3.6 shows how our experiment data was stored on the cloud.

The key generation, encryption and decryption algorithms were implemented using the Scarab homomorphic cryptography library [Perl, 2011]. This C library is the implementation of the Smart and Vercauteren [Smart and Vercauteren, 2010] homomorphic encryption scheme. Other C libraries were also used to store and process large integers such as the GNU Multiple Precision Arithmetic Library [GNU, 2014] and the Fast Library for Number Theory [Hart, 2013]. Our model and these libraries were run on a machine having 4 Gigabytes of memory and 2.4 Gigahertz quad core processor with Ubuntu Linux as the operating system.

Homomorphic public and private keys were generated on the client side. We measured the time required to generate the keys as well as the encryption and decryption times on the client, while homomorphic addition operations happened on the cloud. Table 3.7 shows these measurements. From this table, it can be seen that key generation took more time to execute than other operations because the key generation algorithm had to search for appropriate random numbers to construct the keys. Encryption and decryption times required to compute over 20-bit numbers were within reasonable limits. Although a homomorphic addition over 20-bit numbers took more time than encryption and decryption, this operation would be done on the cloud, which has far more computing resource than the client side.

Smart Meter ID	Timestamp	Encrypted Readings
SM001	03:00am 16/04/2014	dikY3kdkzos322354
SM566	19:15pm 17/05/2014	ieosILC23Kslskeisld
SM019	21:30am 19/10/2013	802edisEIWDkdisle
SM005	07:45pm 20/06/2014	987122kdiesk8392
SM302	12:15am 19/12/2013	powodie87349874d

Table 3.6: Encrypted Smart Meter Data stored on The Cloud

Operation	Time
Key Generation (client)	From 5 to 25 (seconds)
Encryption (20 bits, client)	244.8 (milliseconds)
Decryption (20 bits, client)	314.4 (milliseconds)
Homomorphic Addition (20 bits, cloud)	65.6 (milliseconds)

Table 3.7: Time measured for key generation, encryption, decryption and homomorphic addition operations

3.4.2 Homomorphic Billing Computation in Different Billing Periods

In this experiment, we measured the time required to calculate electricity bills for a household in different billing periods such as in a fortnight, a month and every three months. Assuming that the smart meter would send a reading to the cloud every 15 minutes, we calculated the number of homomorphic additions that would happen on the cloud corresponding to each period. Table 3.8 presents the number of homomorphic additions corresponding to each period. The grid operator would first send a billing request to the cloud, containing the smart meter ID and the time period in which the bill must be calculated. The cloud would use the smart meter ID to identify all the readings recorded by that meter and use the timestamps to retrieve appropriate encrypted readings to perform homomorphic addition operations. Finally, the cloud would send encrypted results back to the grid operator. Table 3.8 also shows the time required by the cloud to perform the number of homomorphic addition operations corresponding to each billing period.

3.4.3 Homomorphic Billing Computation for Different Numbers of Smart Meters in a Fixed Billing Period

In the second experiment, we measured the execution time of the homomorphic addition operations performed by the cloud to calculate the total electricity usage for different numbers of smart meters in one week, assuming that a smart meter sends four readings to the cloud in

Billing Period	Number of Additions	Time (second)
Fortnightly	1343	88.1
Monthly	2879	188.86
Quarterly	8639	566.72

Table 3.8: Time taken to perform homomorphic addition operations to calculate total electricity usage for a smart meter in different billing periods, assuming that a smart meter sends four readings to the cloud in an hour.

Number of Meters	Number of Additions	Time (second)
10	6710	440.18
50	33550	2200.88
100	67100	4401.76

Table 3.9: Time taken to perform homomorphic addition operations to calculate the total electricity usage for different numbers of smart meters in one week, assuming that a smart meter sends four readings in an hour.

an hour. In this experiment, we used a different number of smart meters, i.e. 10, 50 and 100 meters each time the experiment is run. Table 3.9 shows the numbers of smart meters used and the corresponding number of homomorphic addition operations required to calculate the bills for each group of meters in one week. For each group of smart meters, the cloud would find all identification numbers of the meters in the group, read the timestamps to retrieve appropriate encrypted readings and perform homomorphic addition operations on these encrypted data. Table 3.9 also shows the time required by the cloud to perform the number of homomorphic addition operations corresponding to each group of smart meters in one week. From this table, it can be seen that the more meters involved in a computation, the more time it will take. The cloud can speed up the homomorphic computation by using caching. Specifically, it can store the encrypted result of a homomorphic computation of two readings and reuse that encrypted result later when the same timestamps of those encrypted readings appear again in a computing request.

3.5 Performance of Our Algorithm on Multi-core Cloud Servers

Our research work can be extended by creating a parallel version of our algorithm so that the calculation of total energy usage on homomorphically encrypted data can be performed

more efficiently on multi-core cloud servers. In the experiments described previously, the total energy usage was calculated by adding a series of homomorphic additions sequentially on encrypted numbers. This method is not efficient because only one addition can be performed at a time. The computation time can be reduced by applying an algorithm that can compute the homomorphic sum in parallel. In this research, we use a parallel addition algorithm described by Blelloch et al. [Blelloch and Maggs, 2010]. Suppose that the cloud needs to calculate the sum of a sequence S of n homomorphically encrypted numbers:

$$S[1] + S[2] + \cdots + S[n]$$

The computation can be performed in parallel by pairing and adding each element of S having an even index with the next element of S having an odd index, i.e. S[0] is paired with S[1], S[2] with S[3], and so on. The result is a new sequence of $\lceil n/2 \rceil$ numbers that sum to the same value as the sum that we wish to compute. The pairing and summing stage is repeated until, after $\lceil \log_2 n \rceil$ steps, producing a sequence consisting of a single value which is the final sum. Hence, applying this algorithm allows the execution time to be reduced significantly because many addition operations can be done in parallel. Furthermore, we want to determine how much of the computing workload can be done in parallel, especially when we increase the number of terms in a homomorphic addition expression. We demonstrate our point with the following example in which we want to add a homomorphic expression with eight terms:

$$\begin{aligned} & x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 \\ &= x_{12} + x_{34} + x_{56} + x_{78} \\ &= x_{1234} + x_{5678} \\ &= X \end{aligned}$$

Assume that one homomorphic expression takes a time t to complete, then the above expression will take time $7t$ to complete when the computing operations are done sequentially, i.e. one homomorphic addition at a time. However, when applying the Blelloch parallel addition algorithm described above, many homomorphic additions can be executed in parallel, for example, $x_{12} = x_1 + x_2$, $x_{34} = x_3 + x_4$ or $x_{1234} = x_{12} + x_{34} \ldots$. Hence, the total time required to calculate X is $3t$, rather than $7t$ as in the sequential case. We write a program to repeat the calculations above with an increasing number of terms in the homomorphic expression. Table 3.10 shows our results. From this table, when the number of terms in a homomorphic expression increases, i.e. 10, 50, 100, 500 and 1000, the fraction of computing operations that can be performed in parallel also increases significantly, i.e. 55.56%, 87.76%, 92.93%, 98.2%, 99%.

Table 3.10: When the number of terms in a homomorphic expression increases, the fraction of homomorphic addition workload that can be performed in parallel also increases significantly. The symbol t represents the time required to complete one homomorphic addition operation.

Number of Terms in a Homomorphic Expression	Time required (serial case)	Time required (parallel case)	Fraction of Computing Performed in Parallel (%)	Fraction of Computing Performed Sequentially (%)
10	$9t$	$4t$	55.56	44.44
50	$49t$	$6t$	87.76	12.24
100	$99t$	$7t$	92.93	7.07
500	$499t$	$9t$	98.2	1.8
1000	$999t$	$10t$	99.00	1.00

When there are 1000 terms in a homomorphic addition expression, almost all of the computing workload can be done in parallel according to the algorithm we described previously. However, in reality, how much of the computing workload can be executed in parallel also depends on the number of computing threads which are allocated for that operations.

Next, we want to measure how the aforementioned parallel homomorphic addition algorithm can improve the efficiency of the computation of the total energy usage on encrypted smart meter data, especially when the algorithm is run on a multi-core cloud server. To quantify the performance enhancement, we use a metric called speedup, described in [Hennessy and Patterson, 2012]. Speedup is defined by the following formula:

$$S = \frac{T_{old}}{T_{new}}$$

in which, S is the resultant speedup, T_{old} is the old execution time before any improvement is made to the algorithm, for example, when using the sequential homomorphic addition algorithm, and T_{new} is the new execution time after improvements are made over an algorithm, for example, when the homomorphic addition algorithm is parallelized.

Furthermore, according to Amdahl's law [Rodgers, 1985], the speedup S of an algorithm running on a multi-core server depends on the number of threads of execution, and most importantly, on the sequential fraction of the algorithm. Specifically, let $n \in \mathbb{N}$ be the number of threads of execution and $B \in [0, 1]$ be the fraction of the algorithm that is strictly serial, then the time $T(n)$ the algorithm would take to finish when running on n threads of execution is calculated by the following formula:

$$T(n) = T(1)(B + \frac{1}{n}(1 - B))$$

Table 3.11: The speedups achieved when the parallel fraction of the homomorphic addition algorithm is executed by some common processors having different number of cores and threads. These processors are usually found in desktop computers and cloud servers. These results coming from parallel performance calculations with a varying number of terms in a homomorphic addition expression, i.e. 50, 100, 500, 1000 terms as shown in Table 3.10. The corresponding maximum theoretical speedups are also shown for comparison purpose.

Processor	**Cores**	**Threads**	**The computed speedup when using different numbers of terms in a homomorphic expression**			
			50 (terms)	**100 (terms)**	**500 (terms)**	**1000 (terms)**
Intel Core Solo Processor U1500	1	1	1	1	1	1
Intel Core i3-4030U Processor	2	4	2.93	3.30	3.80	3.88
Intel Xeon Processor E5-2630	6	12	5.11	6.75	10.02	10.81
Sun Microsystems UltraSPARC T2	8	64	7.35	11.73	29.99	39.26
Maximum Theoretical Speedup			8.17	14.14	55.56	100

Hence, the speedup is calculated as:

$$S(n) = \frac{T(1)}{T(n)} = \frac{T(1)}{T(1)(B + \frac{1}{n}(1 - B))} = \frac{1}{B + \frac{1}{n}(1 - B)}$$

Amdahl's law also allows us to find the maximum speedup despite the maximum number of threads available. The maximum theoretical speedup is calculated by letting n, the number of threads of execution, go to infinity, as in the following equation:

$$\lim_{n \to \infty} S(N) = \lim_{n \to \infty} \frac{1}{B + \frac{1}{n}(1 - B)} = \frac{1}{B}$$

However, the number of threads of execution n is dependent on the number of cores of a server's Central Processing Unit (CPU). Only one thread can be served by one core at

a time and the CPU will switch between threads if the number of threads generated by a program is larger than the number of cores of the CPU. In the year 2002, Intel corporation introduced a technology called Hyper-Threading in its Xeon server processors and Pentium 4 desktop CPUs, allowing two threads to be run per core. Table 3.11 shows the speedup that we calculated when the parallel computation is performed by many processors having different number of cores and threads. This table demonstrates how an increase of speedup is determined by the size of the homomorphic expression as well as the number of threads used for parallel computation. Because of a limited number of threads used for parallel computation task, the actual speedup is always smaller than the maximum theoretical speedup. From the table, the maximum theoretical speedup when parallelizing the homomorphic addition of 1000 terms in an expression is 100 times. However, if 64 threads are used for such parallel homomorphic addition, then the actual speedup is 39.26 times. This speedup will get closer to the maximum when a larger number of threads is used.

3.6 Conclusion

In this research, we have designed and implemented a secure cloud-based data storage and processing model which can preserve user privacy and the confidentiality of smart meter data on a smart grid. Our research ensures that any smart grid, which we assume to be an Internet of Things with many components connected directly to a public network, can fully benefit from various cloud storage and processing services. This is made possible by our homomorphic computing model using a homomorphic asymmetric key cryptosystem to encrypt data, allowing the cloud to perform most of the computing works directly on encrypted data, specifically, the calculation of customer bills based on the aggregation of encrypted smart meter readings using fixed-point number arithmetics. With practical data from the Smart project, we have done many experiments to estimate the number of homomorphic additions to be performed on the cloud and measured the computation time in various billing periods. Our experiments show several factors that can influence the homomorphic computation time on the cloud such as the length of a billing period, the number of meters involved, or directly by the number of homomorphic addition operations. We also propose a parallel version of our billing algorithm to utilise the processing capacity of multi-core processors in cloud servers, reducing computation time by about 50 percent compared to our sequential algorithm. In the future, we will work on more efficient methods to allow the cloud to further reduce the homomorphic computation time as well as more efficient and scalable cloud computing services on encrypted data.

Chapter 4

Secure Pricing Comparison Model Using Homomorphic Cryptography

Abstract

This chapter describes how we have applied homomorphic encryption in designing and implementing a model to perform secure computation, comparison and integrity checking in a common e-commerce scenario. Specifically, there is a client who wants to perform a transaction with an untrusted cloud-based server which performs most of the computing operations without disclosing the client's financial information from multiple bank accounts. We also design a new mechanism based on homomorphic encryption to minimise the use of a trusted third party and allow client and server to validate transaction outcomes directly in a reliable and secure manner. The implementation our of model is efficient, scalable and can easily be extended to allow an arbitrary number of parties to join the homomorphic cloud-based computation.

4.1 Introduction

In this chapter, we continue our research with another application of homomorphic cryptography to address the computing and security requirements of providing cloud-based data storage and processing capability. As we have mentioned in the introduction of this thesis, one solution to secure the storage and transmission of cloud-based data is the use of traditional cryptographic primitives such as Advanced Encryption Standard [Aes, 2001] or RSA [Rivest et al., 1978b], which are symmetric and asymmetric cryptographic schemes respectively. However, these techniques make it difficult for processing tasks such as searching, updating or

checking the integrity of encrypted data without asking clients to download and decrypt large amounts of data from the cloud. To realise the full potential of cloud computing, better cryptographic schemes are required. They should enable the cloud to perform various computing operations on encrypted data and return encrypted results to customers. Another desirable feature is how a cryptographic scheme can allow different parties to combine their encrypted data and perform some computing tasks on the cloud without compromising the confidentiality and privacy of the data of each party.

The work we present in this chapter will again demonstrate how homomorphic encryption can protect data privacy in cloud-based computations, especially when the computing entity can not be trusted. Ideally, there is a commonly trusted entity to process inputs from all parties involved in a computing operation. Privacy is guaranteed because the trusted entity is supposed to keep the input secrets. However, there are scenarios when the parties performing the computing operations can not be trusted. Specifically, there are cases where the processing entity is semi-honest or even malicious. In the former case, the computing entity is required to follow the protocol, but are permitted to record all data collected during the execution of the protocol. At the end of a computing session, the entity can analyse the data collected in attempt to extract more information than just their inputs and the output of the function. Furthermore, when the computing entity is malicious, many unexpected behaviours can happen when the computing protocol is executed. For example, aborting the protocol can be used as a technique to attempt to learn the inputs and tamper with the final results [Paulet, 2013].

Our Contributions. This chapter describes how we have applied homomorphic encryption in designing and implementing a model to perform secure computation, comparison and integrity checking in a common e-commerce scenario shown in Figure 4.1. In this scenario, the Client is an ordinary Internet user who wants to participate in an e-commerce transaction. Specifically, there is an item listed on a website at a particular price that the Client wants to buy. In a typical online purchase like this, the Client will need to add his favourite item to the shopping bag, then check out and enter his card numbers and other personal details such as phone numbers and address to buy this product. One major disadvantage of this process is that Client has no control over their personal details once they have sent those details to the e-commerce website. Furthermore, some e-commerce websites want to check if the client has sufficient fund before approving the purchase. Therefore, a new method is required which should allow the client balance to be compared with the actual item cost without compromising the confidentiality and privacy of the client. Our research work will address the security and privacy requirements in such scenarios through the following contributions:

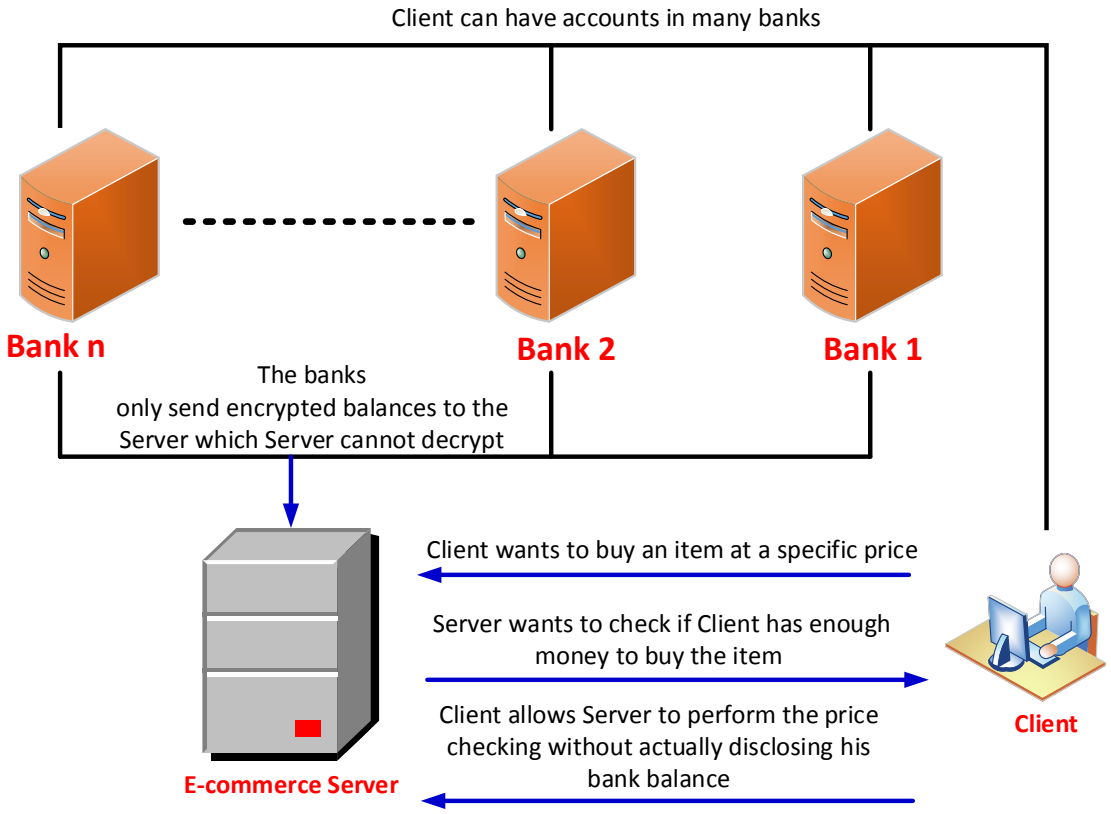


Figure 4.1: An overview of our model.

- We apply homomorphic encryption in a new way to create a cloud-based model to compute directly on encrypted integers. Specifically, our model allows homomorphic addition and comparison to be performed over encrypted binary digits. Data security and client privacy are ensured because most of the computing operations are performed on encrypted data.

- We design a new mechanism based on homomorphic encryption to minimise the use of a trusted third party and allow Client and Server to validate transaction outcomes in a reliable and secure manner.

- The implementation our of model is efficient and powerful because the cryptographic scheme we use has small ciphertext and key size with all homomorphic computing operations performed directly on encrypted binary numbers. Our computing model is scalable and can easily be extended to allow an arbitrary number of parties to join the homomorphic cloud-based computation.

4.2 Background and Related Works

There have been many research works which propose methods to secure online payment system and preserve the privacy of users by means of cryptography and other technologies. In this section, we will provide an overview of some of those methods. Petro et al. [Moreno-Sanchez et al., 2015] proposed a privacy preserving payments in credit networks, which is a protocol with the capability to provide the obliviousness of transactions, thereby offering strong privacy protection for payments. The most prominent feature in their work is the algorithm to compute the maximal credit between two agents, without revealing any information about the credit network, the transaction or the agents themselves. Using an IBM cryptographic co-processor, their model can minimised the cryptographic overhead at the user's end, making the model a good candidate for deployment on mobile devices. However, in their work, the authors only propose and implement the prototype of the algorithm without an actual application of the protocol on mobile devices.

Konidala et al. and his team [Konidala et al., 2012] has proposed a payment security model to protect the confidentiality of customer information when credit card transactions happening on mobile devices. The protected information includes card details, payment amount, transaction time and location so that in cases when systems at the intermediate third-party payment processor/gateway companies are breached, damages to customer privacy and information are minimized. Their model allows customers to use their mobile phones to obtain the merchant's bank account information while being able to instruct their banks to transfer money to the merchant's bank account using public key cryptography and partially blind signature.

A privacy-preserving e-payment scheme was proposed by Ashrafi et al. [Ashrafi and Ng, 2009] to perform authenticity checking and protect the confidentiality of customer sensitive details from various parties involved in an online transaction. The model proposed in their work is a valuable tool to control the information flow in an online transaction, ensuring both communication privacy and information privacy even when an online merchant employs services from third party payment gateways. Their protocol uses a public key encryption and a non-reusable password-based authentication approach by integrating a unique time stamp into the hash all information related to a transaction, thereby preventing many types of replay attacks.

In addition to e-commerce and online purchases, security of transactions are very important in electronic payment systems applied in public transport to replace cash or ticket-based system. The research work proposed by Rupp et al. [Rupp et al., 2015] propose a mechanism

to protect the privacy of customers and the general security of such systems. The authors introduce a fast and efficient cryptographic payment scheme with a detailed framework, security model and implementation. Their proposed system is designed to work under the resource constraints of user devices while meeting real-time computation requirements.

4.3 Secure Homomorphic Price Checking Model

The power and flexibility of the homomorphic computing techniques described in previous sections have allowed us to design a computing framework for our model, which we use to address the Client security and privacy requirement described in the Introduction section. Specifically, as shown in Figure 4.2 the Client wants to buy an item costing X dollars from an e-commerce website. All business transactions are processed by a cloud-based Server to leverage the power and efficiency of cloud computing. The Client has financial accounts $b_1, b_2, \cdots, b_3$ in n independent banks. The Server wants to check if Client has enough money to cover the cost $\$X$ of the item before authorising the purchase. This checking process is accomplished by calculating the total amount of money N of the Client using the formula $N = b_1 + b_2 + \cdots + b_n$ and then performing a comparison between N and X. The Client is allowed to purchase the item only when $N \geq X$. To maximise efficiency, most of the computing operations must be performed on the cloud-based Server while minimising communication with Client. To ensure data security and privacy of Client, there is a major problem we need to address. That is how to perform various computing operations on the cloud-based Server while still preserving the secret of the bank balances $b_1, b_2, \cdots, b_3$ as well as the total amount N. Our contributions can be summarised as follows:

An overview of the technical details of our model is shown in Figure 4.3. We assume that the Client has already been provided with a pair of public and private homomorphic keys generated by using the Smart and Vercauteren [Smart and Vercauteren, 2010] cryptography scheme. In the figure, private and public key is coloured red and green, respectively. The Client's public key is also available to use by all the banks storing the Client's balances as well as any e-commerce Server that the Client communicates with. The Client is interested in buying an item costing $\$X$ dollars and wants to ensure the Server that he has enough money to pay for the items without disclosing the total balance or the balances in all his banks.

To meet all the Client's requirements described above, we have used homomorphic cryptography extensively as well as designing a homomorphic-based protocol to check important results at various stages of the process. From Step 1 in Figure 4.3, it can be seen how the

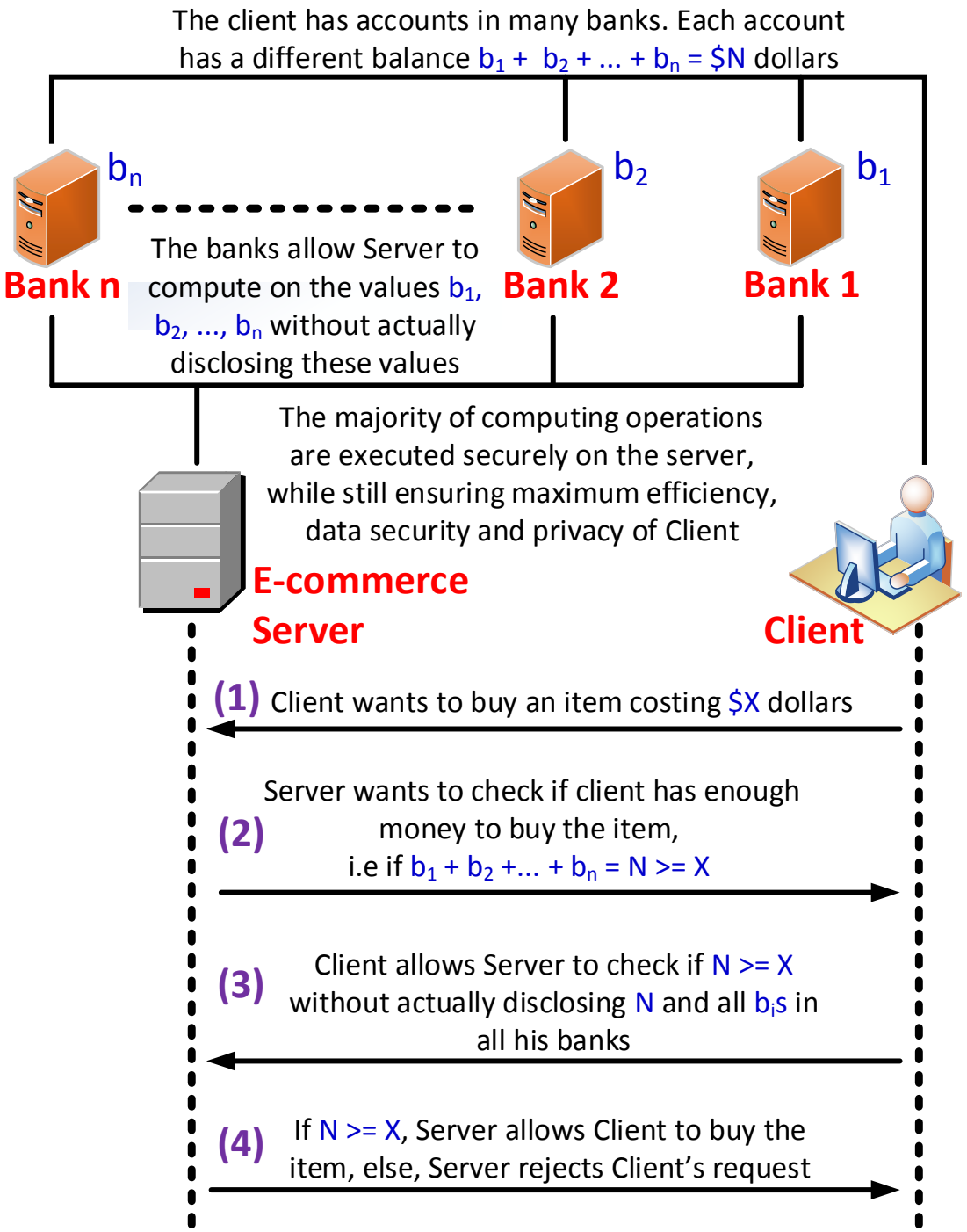


Figure 4.2: An overview of our model.

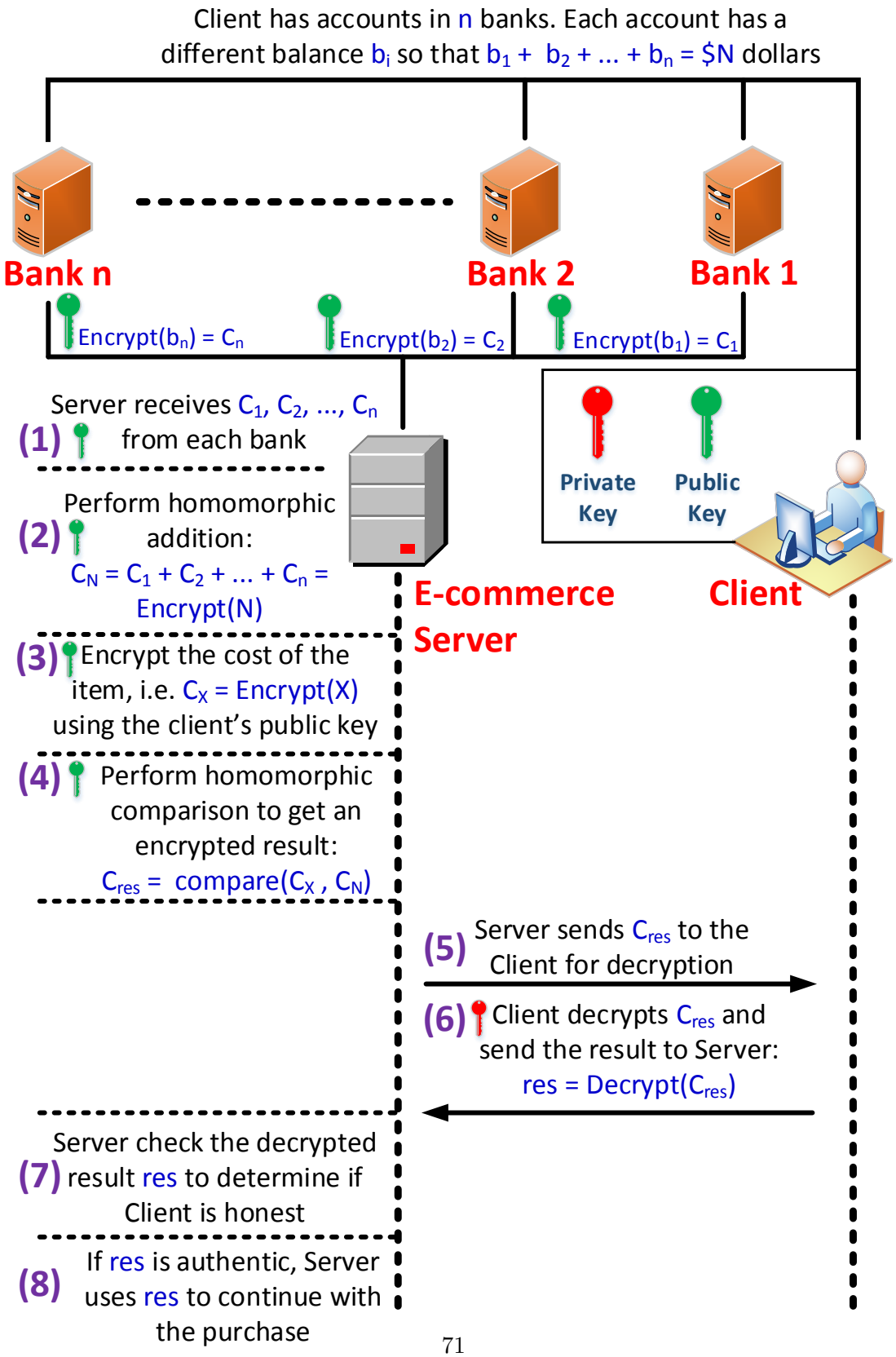


Figure 4.3: A more detailed view of the interactions between Client and Server in our model

Client's balances in each of his n banks, i.e. each of the $b_1, b_2, \cdots, b_n$ is encrypted by using his public key before sending to the Server. Therefore, these balances are still being kept secret from the Server. Each account balance b_i is encrypted bit-by-bit, according to the technique described in Section 3.2.3 to get the following ciphertexts:

$$c_1 = \text{Encrypt}(b_1), c_2 = \text{Encrypt}(b_2), \cdots, c_n = \text{Encrypt}(b_n)$$

In Step 2, Server uses the Client's public key to perform a homomorphic addition of all the c_i ciphertexts obtained from Step 1. The result of this addition is an array C_N of encrypted bits representing the encryption of the Client's total amount of money N. This homomorphic addition operation is carried out according to the theory described in Section 3.2.5. The following equation summarises the process:

$$C_N = C_1 + C_2 + \cdots + C_n = \text{Encrypt}(N)$$

To determine if the Client has enough money to buy the item costing $\$X$ dollars, a comparison operation must be performed to determine whether the Client's total amount of money N is greater than or equal to X, i.e if $N \geq X$. The sum N has already been obtained in encrypted form, i.e. $C_N = \text{Encrypt}(N)$ because of the requirement to protect the Client's privacy. Therefore, the comparison must be performed homomorphically in encrypted domain, requiring the Server to encrypt the item cost X to obtain $C_X = \text{Encrypt}(X)$ using the Client's public key. This process is summarised in Step 3 of Figure 4.3.

One of the main features of the model is shown in Step 4 of Figure 4.3. At this stage, a homomorphic comparison operation is executed in encrypted domain to compare the item cost X and the total amount of money N, both values have been encrypted as C_X and C_N, respectively. Based on the work of Kaosar et al. [Kaosar et al., 2012], a homomorphic comparison algorithm is developed. Details of this algorithm is described in Table 4.1, which shows the execution steps for both plaintext and encrypted numbers. For this algorithm to work, it is assumed that each integer is encoded using the two complement format and each bit of the input integers is encrypted using a homomorphic cryptography algorithm. In our model, the result is an encrypted bit:

$$C_{res} = \text{homoCompare}(C_X, C_N)$$

From table 4.1, it can be seen that the result of the comparison algorithm, which is the most significant bit, can be obtained directly on the plaintext domain. However, on an encrypted domain, the result has to be decrypted to determine which is larger among the two encrypted

	Compare Two Plaintext Integers α and β	**Compare Two Encrypted Integers $E(\alpha)$ and $E(\beta)$** (E(): encryption function using public key) (D(): decryption function using private key)
Input	Two plaintext integers α and β: $\alpha = [\alpha_1, \alpha 2, \cdots, \alpha_n]$ $\beta = [\beta_1, \beta_2, \cdots, \beta_n]$ n: number of bits used for binary encoding.	Two encrypted integers $E(\alpha)$ and $E(\beta)$: $E(\alpha) = [E(\alpha_1), E(\alpha 2), \cdots, E(\alpha_n)]$ $E(\beta) = [E(\beta_1), E(\beta_2), \cdots, E(\beta_n)]$ n: number of bits used for binary encoding.
Steps	**Step 1**: find the binary negation of β: $\bar{\beta} = \text{XOR}(\beta, 2^n - 1)$ **Step 2**: find the two complement $-\beta$ by adding $\bar{\beta}$ with bit 1: $-\beta = \text{add}(\bar{\beta}, 1)$ **Step 3**: compute the binary addition of α and $-\beta$: $\gamma = \text{add}(\alpha, -\beta) = [\gamma_1, \gamma_2, \cdots, \gamma_n]$ **Step 4**: extract the most significant bit of γ: $\gamma_1 = MSB(\gamma)$ $= MSB([\gamma_1, \gamma_2, \cdots, \gamma_n])$	**Step 1**: find the binary negation of $E(\beta)$ using homomorphic XOR: $E(\bar{\beta}) = \text{homoXOR}(E(\beta), E(2^n - 1))$ **Step 2**: find the encrypted two complement $E(-\beta)$ by adding $E(\bar{\beta})$ with encrypted bit E(1): $E(-\beta) = \text{homoAdd}(E(\bar{\beta}), E(1))$ **Step 3**: compute the homomorphic binary addition of $E(\alpha)$ and $E(-\beta)$: $\gamma = \text{homoAdd}(E(\alpha), E(-\beta))$ $= [E(\gamma_1), E(\gamma_2), \cdots, E(\gamma_n)]$ **Step 4**: extract the encrypted most significant bit of $E(\gamma)$: $E(\gamma_1) = EMSB(E(\gamma))$ $= EMSB([E(\gamma_1), E(\gamma_2), \cdots, E(\gamma_n)])$
Output	$\alpha \geq \beta$ if $\gamma_1 = 0$ $\alpha \leq \beta$ if $\gamma_1 = 1$	$\alpha \geq \beta$ if $D(E(\gamma_1)) = 0$ $\alpha \leq \beta$ if $D(E(\gamma_1)) = 1$

Table 4.1: Details of the integer comparison algorithm both in plaintext and encrypted formats

integer inputs. This is the reason why in Step 5 of Figure 4.3, Server has to send C_{res} to Client for decryption. The homomorphic encryption scheme has ensured that Server can execute the comparison without actually knowing the plaintext value of C_N. Furthermore, the result of the comparison is only one encrypted bit, which can only be the number 1 or 0. This information is only sufficient for Server to determine whether Client has enough money to buy the item, i.e. whether $N \geq X$, so that Server can take an appropriate action as described in Step 7 and 8 of Figure 4.3. However, before Server can proceed with Step 7 and 8, it must have a mechanism to check whether the decryption value sent from Client is actually the decrypted result of the homomorphic comparison operation. This problem will be addressed in the next section.

4.3.1 Checking Client Decryption Results

In our model, almost all the computing operations are performed on encrypted data by the untrusted cloud-based e-commerce Server, from aggregating the bank balances to calculating the total amount and comparing that value with a fixed price. Furthermore, just like any other application processing encrypted data, at some stage in the computing process, some of the encrypted data must be decrypted to compute the final result, provided that the decrypted values do not compromise the security of the whole computing model. That is the purpose of Step 5 and 6 of our model in Figure 4.3. However, there is an important obstacle that Server has to overcome: checking whether the decryption value sent from Client is actually the decrypted result of the homomorphic comparison operation. There is a strong motivation for such checks. The Client can change a decrypted value to his favourite outcome before returning that value to Server, compromising the reliability of our computing model. Therefore, we have developed a solution using the unique characteristics of homomorphic encryption to provide Server with a reliable way to ensure that the decrypted result returned by Client is actually the decrypted result of the homomorphic comparison operation.

Our algorithm is designed based on the homomorphic XOR (homoXOR) and homomorphic AND (homoAND) operations of the cryptographic scheme which has been using to perform all the computing operations on the server. We also assume that Server has access to the public key of Client to encrypt extra information necessary for the checking process. The algorithm comprises several steps which are described as followed:

- **Inputs and functions used**:
  - C_{res} = compare(C_X, C_N), an encrypted bit, which is the encrypted result of the homomorphic comparison operation between C_X and C_N.
  - S: a security parameter which is also the number of binary digits of other parameters used in later steps of the algorithm
  - E() is the homomorphic encryption function, accepting the Client's public key and one plaintext binary digit as inputs and return a ciphertext.
  - D() is the homomorphic decryption function, accepting the Client's private key and one ciphertext as inputs and return a plaintext binary digit.
  - homoAND() and homoXOR() are the two homomorphic operations, each of them takes as inputs two encrypted binary digits and the Client's public key and out-

puts an encrypted bit corresponding to the results of the AND and XOR operation running on plaintext bits, respectively.

- **Step 1**: Server encrypts two bits, 0 and 1 using the public key of Client:

$$C_0 = E(0) \text{ and } C_1 = E(1)$$

- **Step 2**: Server chooses a secret S-bit random number M and encrypts each bit using the public key of the Client:

$$M = [m_1, m_2, \cdots, m_S] \longrightarrow E(M) = [E(m_1), E(m_2), \cdots, E(m_S)]$$

- **Step 3**: Server creates a list of homomorphic operations having S elements. Each element in the list is either homoAND() or homoXOR() function selected randomly. The list is called *homoOpList* and can be summarised as followed:

$$\begin{aligned} \textit{homoOpList} &= [[\text{homoAnd() or homoXOR()}]_1, \\ &= [\text{homoAnd() or homoXOR()}]_2, \\ &= \cdots, \\ &= [\text{homoAnd() or homoXOR()}]_S] \end{aligned}$$

- **Step 4**: Server creates another list of ciphertexts called *secondOpList* which has S elements. Each element in this list is randomly selected from the set $\{C_0, C_1, C_{res}\}$:

$$\textit{secondOpList} = [[C_0|C_1|C_{res}]_1, [C_0|C_1|C_{res}]_2, \cdots, [C_0|C_1|C_{res}]_S]$$

- **Step 5**: For each of the encrypted bit of $E(M)$, Server selects the corresponding ciphertext in *secondOpList*, either C_0, C_1 or C_{res}, and uses the corresponding homomorphic operation in the list *homoOpList*, either homoAND() or homoXOR() to compute on these two encrypted bits, producing a new list of ciphertext called the *resultList*. This process is shown in Table 4.2.

  For every transaction with a particular client, after finishing this step, the server will save the following information:

  - The secret S-bit random number M created in Step 2 in plaintext format:

$$M = [m_1, m_2, \cdots, m_S]$$

Encrypted S-bit number $E(M)$	$E(m_1)$	$E(m_2)$	$\cdots$	$E(m_S)$
homoOpList	homoAND() or homoXOR()	homoAND() or homoXOR()	$\cdots$	homoAND() or homoXOR()
secondOpList	$[C_0 \vert C_1 \vert C_{res}]_1$	$[C_0 \vert C_1 \vert C_{res}]_2$	$\cdots$	$[C_0 \vert C_1 \vert C_{res}]_S]$
resultList	C_{res1}	C_{res2}	$\cdots$	C_{resS}

Table 4.2: Server computes extra ciphertexts based on the encrypted result of the homomorphic comparison operations. These ciphertexts are used to check the decrypted comparison result sent from Client.

  - The list of random homomorphic operation used, i.e. the *homoOpList* created in Step 3, in which each element contains either homoAND() or homoXOR() function.
  - The list of ciphertexts used as second operands for each of the homomorphic operation, i.e *secondOpList* created in Step 4, in which each element is randomly selected from the set $\{C_0, C_1, C_{res}\}$.

- **Step 6**: Server sends the a decryption request to Client together with the following ciphertexts: C_{res}, which is the 1-bit decrypted result of the homomorphic computation, and all the ciphertexts in *resultList*, i.e. C_{res1}, $C_{res2}, \cdots, C_{resS}$.

- **Step 7**: Client receives and decrypts the ciphertexts and sends back to Server:

$$\begin{aligned} \text{bitRes} &= D(C_{Res}) \\ \text{bit}_1 &= D(C_{Res1}) \\ \text{bit}_2 &= D(C_{Res2}) \\ &\cdots\cdots\cdots\cdots \\ \text{bit}_S &= D(C_{ResS}) \end{aligned}$$

- **Step 8**: Server wants to check to make sure that bitRes $= D(C_{Res})$. It would mean that Client has been honest and sent correct decrypted bits to the Server. The computing operations performed by Server are shown in Table 4.3.

- **Step 9**: If all the following statements are true:

  Server concludes that bitRes $= D(C_{Res})$, meaning Client has been honest and sent the correct decrypted result of the homomorphic comparison. Server can now rely on C_{Res} to continue on with the transaction.

Secret S-bit random number M	m_1	m_2	...	m_S
Plaintext operation corresponding to each operation in homoOpList	(XOR | AND)	(XOR | AND)	...	(XOR | AND)
Plaintext bit corresponding to each ciphertext in secondOpList	(0 | 1 | bitRes)	(0 | 1 | bitRes)	...	(0 | 1 | bitRes)
Result Bits	checkBit_1	checkBit_2	...	checkBit_S

Table 4.3: Server performs a series of computing operatios to check if Client has not altered any decrypted bit, especially the result of the homomorphic comparison, before sending them back to Server.

$$\begin{aligned} \text{checkBit}_1 &= \text{bit}_1 \\ \text{checkBit}_2 &= \text{bit}_2 \\ &\ldots\ldots\ldots\ldots\ldots\ldots \\ \text{checkBit}_S &= \text{bit}_S \end{aligned}$$

If one of the above statements are false, Server concludes that Client has changed at least one decrypted bit. It can then cancel the transaction or ask Client to send the decrypted results again and executes all the steps above from the beginning.

4.4 The Security of Our Model in Various Scenarios

The homomorphic encryption scheme used in our model can only encrypt binary numbers. Therefore, the possible homomorphic operations in encrypted domain are mapped directly to typical operations that can be applied to binary numbers. Specifically, there are two homomorphic operations that can be performed over encrypted bits. The first one is the **homoAND** operation which has the same effect on two encrypted bits as the **AND** logical operation over two plaintext bits as shown in the following equation:

$$D(\text{homoAND}(E(b1), E(b2)) = D(E(\text{AND}(b1, b2)) = \text{AND}(b1, b2)$$

The equation above means that applying the **homoAND** over the ciphertexts of two encrypted bits, i.e. $E(b1)$ and $E(b2)$, will produce another ciphertext which can be decrypted to get the result of applying the **AND** operation over the corresponding plaintext bits $b1$ and $b2$.

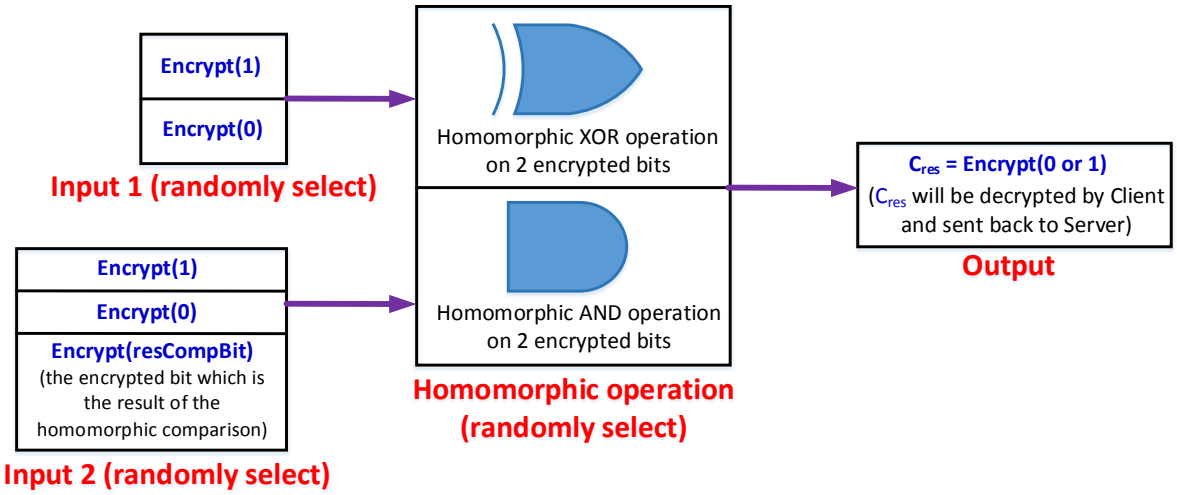


Figure 4.4: An overview of how an encrypted check bit is calculated using random encrypted inputs and random homomorphic XOR or AND circuit

The second homomorphic operation is **homoXOR** in encrypted domain mapping directly to **XOR** in plaintext domain. It can be described in the following equation:

$$D(\text{homoXOR}(E(b1), E(b2)) = D(E(\text{XOR}(b1, b2)) = \text{XOR}(b1, b2)$$

Applying **homoXOR** over two encrypted bits $E(b1)$, $E(b2)$ has the same effect as applying **XOR** over two plaintext bits $b1$ and $b2$.

These two homomorphic operations require input ciphertexts and the public key only, without needing to perform any decryption during the computing process. Therefore, these homomorphic operations do not compromise data security when computing the outputs. Furthermore, as we have already shown, these operations can be used to compute more complex results securely, such as addition, subtraction, multiplication of encrypted integers or performing secure comparison of two integers in encrypted domain.

Next, we describe in detail how our model can protect the security of data and privacy of clients under different application scenarios. These cases can be typical attack scenarios in which there are deliberate acts of the participating parties to violate the secure steps specified in our protocol or they can be absolutely unexpected scenarios caused by neither the client, the e-commerce server or the bank server. In each case, we will first summarise the scenario including all the details related to the possible security violation. Then we prove how the built-in features of our protocol can prevent such types of attacks from happening. Finally, we suggest all possible enhancements that can be applied to our model in each scenario.

- Scenario 1: A client attempts to change decryption results before sending to the server

  In our model, the server has to rely on a decryption sent from the client to determine the exact outcome of the homomorphic comparison. However, there is always a possibility that clients may change decryption results, making it necessary to find a way for the server to check the decryption result using the encrypted data already available and the client's homomorphic public key. This is a motivation for us to propose a method using both **homoAND** and **homoXOR** operations to securely check a decrypted result sent from the client. This approach is based on the fact that given the output, i.e 0 or 1, of a logical expression involving either **AND** or **XOR**, it is difficult to determine exactly the 2 input bits and the logical operation that has been used. When given multiple of such expressions, even if someone can guess exactly one of them, i.e exact input bits and what logical operation has been used, the probability of guessing correctly all expressions will decrease significantly when a larger number of logical expressions is used. Therefore, our checking model not only requires the client to decrypt the homomorphic comparison result, but also requires the client to decrypt the outcomes of a few more homomorphic expressions. Each of them has the format shown in Figure 4.4. The first input of this homomorphic expression is an encryption of 1 or 0 which will be chosen randomly by the server and encrypted by the homomorphic public key. The second input is also chosen at random from three ciphertexts: the homomorphic comparison result or the encryption of 0 or 1. A homomorphic operation is also selected at random, i.e. either **homoAND** or **homoXOR**, to combine the two inputs and produce an encrypted output, which is sent to the user to decrypt. When a user receive a decryption request, what he can see is a list of ciphertexts to be decrypted. He does not know whether a ciphertext is the encrypted result of the comparison or the check expression. After using his homomorphic private key to decrypt these ciphertexts, he can only see a list of binary digits. If the user changes at least one bit in this list, the server will be able to detect that change using our algorithm described in the previous section.

- Scenario 2: Checking the integrity of all messages sent between server and client.
  Integrity checking is an indispensable part of any secure protocol, including the one we propose in this research work. In order to successfully decipher all messages and make a decision whether to authorise a transaction, both server and client needs to be certain that all messages they send and receive are intact on their ways along the

communication lines. These messages can be compromised because of many reasons, either from unexpected technical problems caused by a faulty communication system to deliberate attempts of malicious hackers to intercept and tamper with or change the sequence of these messages. To deal with such situations, we can implement an integrity checking mechanism to supplement the homomorphic encryption module. This integrity check is achieved by allocating a sequence number to each message sent and received by both client and server. This number, however, is kept in sync by both the client and server rather than attached directly to each message. For each message to be sent, we hash the concatenation of the content of the message and the sequence number and encrypt the hash together with the body of the message to be sent. Using this method, both the server and client always know next sequence number to receive and they can decrypt and compare the hash for integrity checking. Therefore, if a message is intercepted or the order of messages is changed along the transmission line, the client and server can detect such changes immediately and can cancel the transaction or request messages to be resent.

- Scenario 3: A man-in-the-middle attack happens in which a malicious user intercepts the communication between the server and client and replays all those messages at a later time

  While a sequence number and message authentication code created from a hash function can prevent tampering with the order of encrypted messages, there is still a possibility that a malicious hacker can record all the messages in one session between a client and a server. These messages can be used later in a replay attack so that a client or server are made to believe that it is receiving new information from another party while in fact, it is receiving the old information from an attacker. Therefore, our model needs to have a mechanism to uniquely identify each homomorphic session, enabling the participating party to quickly detect and avoid replay attacks. In addition to the traditional methods that allow us to achieve such functionality, such as using public key encryption and Secure Socket Layer (SSL) connections, we can implement other methods such as sending random numbers at the beginning of each session so that both client and server can check the hash of those numbers to detect a replay attack when the hash results are inconsistent.

4.5 Implementation and Experiments

To implement our model, we have built a software prototype using the C programming language to simulate all the steps described in our model. Because this cryptographic scheme is used to encrypt binary data, we assume that all inputs are discrete numbers that can be converted to integers. Encrypting an input means that each bit of its binary representation is encrypted using the public homomorphic key. Likewise, decrypting an output of a homomorphic function requires each ciphertext to be decrypted separately using the homomorphic private key to obtain an array of binary digits, which will subsequently be converted to an integer. In our implementation, we wrote several software modules to convert integers into their corresponding two complement binary representation and back from binary to integers ready for encryption and decryption processes, respectively.

The key generation, encryption and decryption were implemented using the Scarab homomorphic cryptography library [Perl, 2011]. It was written in the C language as an implementation of the Smart and Vercauteren [Smart and Vercauteren, 2010] homomorphic encryption scheme. This open source library was created to provide the developer community and researchers with a platform to build further applications and experiments using homomorphic cryptography. It allows us to focus on building the main functionalities of our model rather than the low level implementation details. In many of our experiments with homomorphic encryption, we needed to experiment with large integers, for example, when creating large prime numbers or when storing results of several multiplication involving large integers. However, the built-in integer type of the C programming language, which has a size limit of 4 bytes on a 32 bits system or 8 bytes on 64 bits system, is not large enough for our experiments. Therefore, we used the GNU Multiple Precision Arithmetic Library [GNU, 2014] (GMP), which is an open source and portable library written in C for arbitrary precision arithmetic on integers. GMP is designed to give good performance for our applications in cases when we needed a few hundred bits of precision by carefully keeping the overhead at a minimum. Other C libraries were also used to handle store and process large integers such as the The Multiple Precision Floating-Point Reliable Library [Guillaume Hanrot, 2013] and the Fast Library for Number Theory [Hart, 2013].

Our model and these libraries were run on a machine having 16 Gigabytes of memory and 3.5 Gigahertz quad core processor with Ubuntu Linux as the operating system. We represented each large integer in memory by a large vector of bits. The encryption process receives an input bit and outputs a ciphertext which is a large integer. Therefore, an encrypted integer

is represented as an array of large integers corresponding to each bit of the two complement binary representation of that integer. To evaluate the performance of our software modules, we write codes to measure the time each task were executed directly in every functions.

In the first experiment, we began with measuring the time taken to generate the public and private homomorphic key pair. This process requires the generation of large prime numbers. A few large random numbers were generated and their primality were checked until all primality tests were satisfied. Therefore, we obtained a small variation in the execution time of the key generation algorithm. Next, we measured the time taken to encrypt and decrypt one bit using the homomorphic key pair that has been generated. Finally, we measured the time taken to execute homomorphic addition and multiplication given two ciphertexts and the public homomorphic key. Table 5.1 shows the results of all these experiments described above.

Homomorphic Operation	Time
Key Generation	From 5 to 25 (seconds)
Encryption	12.24 (milliseconds)
Decryption	15.72 (milliseconds)
Addition (homoXOR)	52.89 (milliseconds)
Multiplication (homoAND)	65.74 (milliseconds)

Table 4.4: Execution time of homomorphic operations

Number of Check Bits	Generation Time (milliseconds)	Probability of a Failed Check
1	65.74	0.5
2	131.48	0.24
4	262.96	0.0625
8	525.92	0.00391
16	1051.84	1.52E-05
32	2103.68	2.32E-10

Table 4.5: Experiments carried out with different number of check bits to measure the probability of a failed check in each case.

In subsequent experiments, we tested our scheme with different number of check bits to determine the optimal number of check bits that a client has to decrypt to ensure the authenticity of the homomorphic comparison result. Figure 4.4 shows that the second operand of each check expression is randomly chosen among three posibilities, i.e. the encrypted bit 1, the encrypted

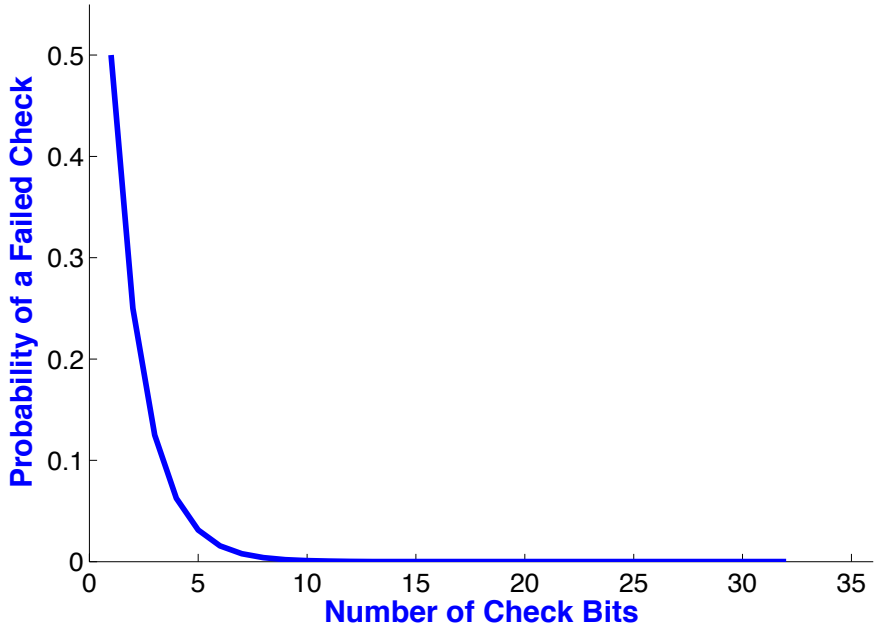


Figure 4.5: The relationship between the number of check bits and the probability of a failed check.

bit 0 or the homomorphic comparison results. However, some of our experiments showed that there were cases when the homomorphic comparison result was not chosen at all because of this random selection. In these cases, the server could not detect whether the homomorphic encrypted result had been changed by the client because it was not used to construct any check expression. Therefore, our testing software modules allowed the server to specify how many times the homomorphic encrypted result was used to construct all check expressions.

For each test case, we used different number of check bits, from 1 to 32 check bits, as shown in Table 4.5. We measured the generation time of those check bits as well as the probability of a failed check, i.e. when the server fails to detect that the client has decrypted and changed at least one bit, whether that bit is the homomorphic comparison result or one of the check bit. This probability of a failed check was also shown in a graph in Figure 4.5. As the number of check bit increases, the probability of a failed check decreases significantly. However, an appropriate number of check bits should be selected so that performance of the model is not affected while maintaining a reasonably low probability of a failed check. This will be a trade-off problem between performance and security that can be determined from the application scenarios.

4.6 Conclusion

In this research work, we have applied homomorphic encryption to create a privacy-preserving price checking model allowing an e-commerce server to check if a client has enough money to buy one of its products without knowing the actual balance of the potential client. This feature is achieved by performing the comparison operation directly on encrypted item cost and encrypted total balance using the public key of a client. We also propose a method for the server to check the decrypted comparison result sent from client. Many experiments were performed with different number of check bits to measure the probability of a failed check, indicating a trade-off between number of check bits and the authenticity of decrypted homomorphic comparison results. In future research works, we will use this approach to build applications that require pubic key holders to decrypt intermediate homomorphic computation results such as secure k-mean clustering.

Chapter 5

Cloud-based Homomorphic Secret Sharing and Access Control

Abstract

Recent improvements in cloud computing technology have provided end users with various ways to outsource their data storage and computing to third party cloud services. Besides the benefits and conveniences of cloud computing, storing and processing data on the cloud without adequate protection and safeguarding mechanisms may affect data security and user privacy, especially when there is a large amount of data coming from many users. In this research work, we propose a cloud-based computing model which provides a business organisation with the capability to outsource its computing needs to a third party cloud service without compromising data security and the privacy of its employees. We use novel features of homomorphic encryption to allow a secret to be shared among multiple parties and later combine on the cloud in encrypted form without revealing the actual shares to the cloud. Homomorphic cryptography allows the cloud to do most of the computing tasks to recreate the shared secret without affecting data security and user privacy. We also show how the proposed homomorphic secret sharing scheme can be applied to create a cloud-based access control application which enables a group of users to control access to encrypted documents storing on a semi-trusted cloud.

5.1 Introduction

In this chapter of the thesis, we propose another novel application of homomorphic cryptography in secure secret sharing. The advancement of cloud computing technologies has provided

users and business organisations with various cloud-based options to store and access information externally, across multiple platforms and geographic locations. Various cryptographic techniques have been used to strengthen information security and protect the privacy of cloud-based data. Furthermore, in large business organisations, the access to highly sensitive data might be controlled by a group of users rather than a single user as shown in Figure 5.1. In this case, members of the group controlling access to those data must have a mechanism to jointly authorise access to encrypted documents stored on the cloud. Shamir's secret sharing scheme [Shamir, 1979] provides a solution to such problem by mathematically dividing the encryption key into multiple independent pieces which are delivered to each member of the group controlling access. However, this scheme requires a trusted party to collect and combine shares of the encryption key before performing computation works to recover the original key. This requirement may not be applicable in cloud-based storage and computing scenarios in which the main computing agents are semi-trusted cloud servers located outside the business organisation. Therefore, a new secure secret sharing model is required to protect the shares while computing and recovering the secret to preserve the privacy of each member of the group controlling access to the encrypted documents.

One solution to secure the storage and transmission of cloud-based data is the use of traditional cryptographic primitives such as Advanced Encryption Standard [Aes, 2001] or RSA [Rivest et al., 1978b], which are symmetric and asymmetric cryptographic schemes respectively. However, these techniques make it difficult for processing tasks such as searching, updating or checking the integrity of encrypted data without asking clients to download and decrypt large amounts of data from the cloud. To realise the full potential of cloud computing, better cryptographic schemes are required. They should enable the cloud to perform various computing operations on encrypted data and return encrypted results to customers. Another desirable feature is how a cryptographic scheme can allow different parties to combine their encrypted data and perform some computing tasks on the cloud without compromising the confidentiality and privacy of the data of each party.

Homomorphic encryption is a very important tool to protect data privacy in cloud-based computations, especially when the computing entity can not be trusted. Ideally, there is a commonly trusted entity to process inputs from all parties involved in a computing operation. Privacy is guaranteed because the trusted entity is supposed to keep the input secrets. However, there are scenarios when the parties performing the computing operations can not be trusted. Specifically, there are cases where the processing entity is semi-honest or even malicious. In the former case, the computing entity is required to follow the protocol, but are permitted

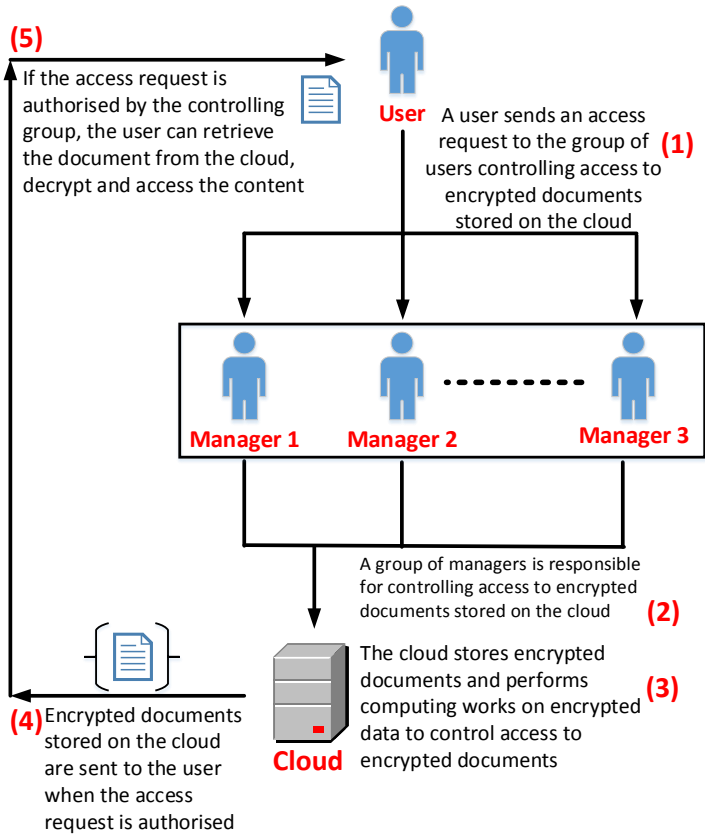


Figure 5.1: Access to encrypted documents stored on the cloud is controlled by a group of users.

to record all data collected during the execution of the protocol. At the end of a computing session, the entity can analyse the data collected in attempt to extract more information than just their inputs and the output of the function. Furthermore, when the computing entity is malicious, many unexpected behaviours can happen when the computing protocol is executed. For example, aborting the protocol can be used as a technique to attempt to learn the inputs and tamper with the final results [Paulet, 2013].

Our Contributions. In this research work, we propose a new computing model which allows users of a business organisation to outsource their data storage and computing tasks to a third party cloud services while still protecting user privacy and the security of their data. We show how novel features of homomorphic cryptography can be applied so that the cloud can do major computing tasks directly on encrypted data and output encrypted results using only the public key. Our model provides details about how a homomorphic key pair is generated and its copies are distributed among members of the organisation for encryption and decryption of

outsourced data. The main focus of our research is how any group of users in the organisation can cooperate to generate a shared secret using the homomorphic computing capability of the cloud. We show how the cloud can do most of the computing task in the process of generating the shared secret without affecting data security and privacy of any user in the group. Our contributions can be summarised as follows:

- We create a homomorphic key distribution model which allows users of an organisation to outsource data storage and computing tasks to the cloud without compromising data security and user privacy.
- We propose a homomorphic secret sharing scheme to help any group of users to create a shared secret with most of the computing task to create, distribute and recreate the shared secret to be done on the cloud. Each user in the group is able to keep his or her share of the secret private while still being able to combine these shares on the cloud.
- We implement an access control model which applies the proposed homomorphic secret sharing scheme to provide a group of users with the ability to control access to encrypted documents stored on the cloud. Most of the computing operations required for each group member to authorise access are done on the cloud over encrypted shares of the secret, therefore the power of the cloud can be fully utilised without affecting the security of all shares of the secret of the controlling party as well as documents stored on the cloud.

5.2 Background and Related Works

5.2.1 Polynomial Interpolation and Its Applications in Shamir' Secret Sharing

Polynomial interpolation is an essential mathematical tool which has been applied in a wide range of computer science research works and applications such as cryptography, signal processing, biometrics and data mining. Interpolation is the process of calculating a function called the interpolating polynomial which passes through a given set of discrete data points. This means that the interpolating polynomial can be used to recalculate the data points exactly, especially, it can estimate any data points in the given range with a high level of accuracy. Complex functions can be simplified in a range by sampling a number of data points in that range and then calculating a simpler interpolating function from those points using various polynomial interpolation techniques.

Given a set of distinct points, the calculated interpolating polynomial is unique. However, this unique polynomial can have many forms which are results of using different interpolation techniques such as direct interpolation, Newton's or Lagrange's method. In this research work, we focus on the Lagrange form of the interpolating polynomial, which is described as follows:

- Given a set of $k+1$ data points: $(x_0, y_0), \ldots, (x_j, y_j), \ldots, (x_k, y_k)$ where no two x_j are the same, the interpolation polynomial in the Lagrange form is a linear combination

$$L(x) := \sum_{j=0}^{k} y_j \ell_j(x)$$

  of many Lagrange basis polynomials, each one of them corresponding to a point in the given set of data points.

- A Lagrange basis polynomial is defined according to the following definition:

$$\ell_j(x) = \prod_{\substack{0 \le m \le k \\ m \neq j}} \frac{x - x_m}{x_j - x_m}$$

$$= \frac{(x - x_0)}{(x_j - x_0)} \cdots \frac{(x - x_{j-1})}{(x_j - x_{j-1})} \frac{(x - x_{j+1})}{(x_j - x_{j+1})} \cdots \frac{(x - x_k)}{(x_j - x_k)}$$

In cryptography, interpolation polynomial in Lagrange form has been used extensively in a research work proposed by Shamir [Shamir, 1979] to share a common secret among a group of users. Each member of the group will have his or her own unique part of the shared secret. An interesting aspect of the Shamir's scheme is that it does not rely on all the group members to combine their shares because that requirement might be difficult to achieve in practice. Instead, even some of the shares are sufficient to recalculate the original secret according to a threshold scheme that Shamir proposed using a proven theorem that a polynomial of degree $k-1$ can be constructed from k data points. Shamir's secret sharing scheme can be summarised in the following steps:

- Randomly select $k-1$ positive integers $a_1, \ldots, a_{k-1}$ and let $a_0 = S$ which is the secret key that we need to share.

- The following polynomial of degree $k-1$ is constructed:

$$f(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{k-1} x^{k-1}$$

- Each data point in the set of n points has the format $(i, f(i))$, in which i can be chosen at random and $f(i)$ is the result obtained by evaluating i using the polynomial shown above. Each member of the group is given one out of n points.

- Because the original polynomial used to generate n points has degree $k-1$, it can be exactly represented by using an interpolating polynomial which goes through k out of n given points. The constant coefficient of the interpolating polynomial is also the shared secret that we need to find.

Shamir's secret sharing scheme has many advantages. It is mathematically robust because it uses Lagrange's polynomial interpolation technique to interpolate a given set of data points and obtain the constant coefficient as the shared secret. Furthermore, Shamir's scheme does not require the participations of all the group member to reconstruct a shared secret. Instead, a threshold can be given so that an administrator can choose the minimum number of shares that are required to reconstruct the shared secret. However, one of the limitations of Shamir's scheme is that the computing agent performing the secre reconstruction work needs to know all the values of k shares in plaintext to perform the computing operations. Therefore, the agent must be trusted, otherwise, it can take copies of the shared secret and perform the secret reconstruction computation without the authorisation of the parties keeping shares of the secret. This limitation can be addressed by encrypting all the shares of the secret and finding a solution to compute and reconstruct the shared secret directly on encrypted data.

5.2.2 Related Works

In this section, we present an overview of various key management schemes which propose either centralised or distributed models to enable a group or subgroups of users to control access and share data. Typically, a complete key management scheme is a complex combination of several modules such as key generation, distribution and access control. Therefore, it is very useful to have an overview of different categories of key management schemes including their principles, advantages, limitations and various scenarios in which these schemes are applied.

The first group of research works propose session key distribution or group key management schemes such as [Wong et al., 2000] and [Aparna and Amberker, 2010]. These schemes provide secure communication for all members by maintaining and updating a secret session key. The second group of key management schemes is called hierarchical access control schemes such as [Crampton, 2009] and [Atallah et al., 2009] which provide secure content access and

communication of groups with hierarchies so that users in higher levels in the hierarchy can access contents at lower levels. Furthermore, the methods used to generate and manage access keys also play an important part in classifying access control schemes. Some schemes such as [Zou and Dai, 2006] and [Guo et al., 2013] manage keys in a centralised way with the use of a trusted party to generate, distribute and monitor dynamic changes in a group. Alternatively, keys can be generated using a decentralised approach with the uniform contributions of all members of the group controlling access to encrypted resources as shown in research works such as [Wu et al., 2009] and [Shimizu et al., 2012].

Although many novel and interesting approaches have been proposed, to the best of our knowledge, there is no research work focusing on access control scenarios when the encryption key is split into multiple pieces and delivered to members of a group controlling access to encrypted documents stored on the cloud. In these cases, when an access request is authorised by the group members, there must be a trusted party to collect, decrypt and combine these pieces to regenerate the access key which is then used to decrypt the requested documents. However, when these storage and computing tasks are outsourced to the cloud, new solutions are require because cloud servers are typically not fully trusted by users, especially when there are sensitive documents stored in encrypted forms. Therefore, in this research, we propose a key management scheme and access control model using homomorphic cryptography to allow the key generation work to be executed on semi-trusted cloud servers, opening new possibilities for applications that can fully utilise the computing power of the cloud while providing maximum protection for data security and user privacy.

5.3 Secure Homomorphic Encrypted Secret Sharing Model

In this research work, we want to create a cloud-based computing model which provides a business organisation with the capability to outsource its computing needs to a third party cloud service without compromising data security and the privacy of its employees. The novel features of the homomorphic computing techniques described in previous sections have allowed us to create such a model, which we will describe in detail in this section.

The key generation and distribution mechanism is illustrated in Figure 5.2. The third-party cloud service is a semi-trusted cloud processing the majority of computing operations required by the organisation. This semi-honest characteristic of the cloud means that the cloud will follow the protocol specified by the business organisation, but it may record and store all data collected during the execution of the protocol to analyse or attempt to extract

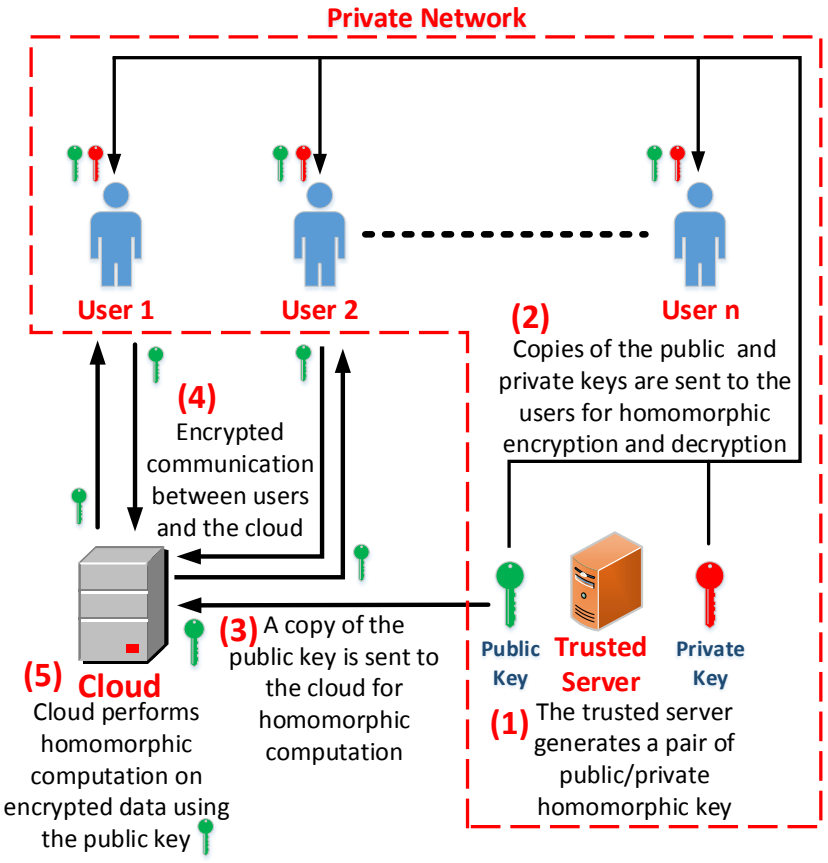


Figure 5.2: The generation and distribution of homomorphic keys to users.

more information for other purposes. Therefore, to preserve data security and user privacy, data must be encrypted before uploading to the cloud for further homomorphic computing operations. We use the Smart and Vercauteren [Smart and Vercauteren, 2010] homomorphic scheme which is an asymmetric cryptography scheme requiring a pair of public and private keys. To maximise the security of our model, these keys are generated by a trusted server protected by the organisation as shown in Figure 5.2. After the key generation process finishes, the trusted server will send each employee of the organisation a copy of the pair of public and private keys. In any computing scenario involving the cloud, each user will use his or her public key to encrypt the data before sending to the cloud and then use the corresponding private key to decrypt the data sent from the cloud. The users are supposed to keep their private keys secret. The cloud does not have access to the private key. Instead, the cloud is given a copy of the public key, which is sufficient to perform all of the homomorphic computing operation on the encrypted data sent from the users in the organisation.

5.3.1 Homomorphic Computation of a Shared Secret on Cloud

One practical application of our cloud-based homomorphic computing model is the sharing of a common secret among a specific group of users. A secret-sharing scheme is a method by which shares of a secret are distributed only to authorized subsets of parties. To reconstruct the original secret, all authorised parties need to combine their shares together. In cases when it is not possible to combine all participant shares of the secret, threshold schemes like those proposed by Blakley [Blakley, 1899] and Shamir [Shamir, 1979] can be used in which any subset having at least k out of the total n shares is sufficient to reconstruct the original secret. Secret sharing plays a crucial part in various cryptographic and distributed computing applications such as oblivious transfer [Tassa, 2011], secure multiparty computation [Cramer et al., 2000], attribute-based encryption [Goyal et al., 2006], and access control [Naor and Wool, 1998].

In this section, we will describe a protocol which allows any group of users in the organisation to construct a shared secret which is not accessible to any user outside that particular group. Most of the computing operations to construct the shared secret will be done on the cloud using homomorphic cryptography. Therefore, the security of data uploaded to the cloud is guaranteed to be protected by means of encryption, while still allowing the full computing power of the cloud to be utilised. Our protocol is based on Lagrange's polynomial interpolation technique [Quadling, 1966] and Shamir's secret sharing scheme [Shamir, 1979]. Our method is based on the approach proposed by those schemes, which means viewing a shared secret as a polynomial, or more specifically, the shared secret can be a constant coefficient of a polynomial. This secret is shared among a group by having each member of the group keeping one point (x, y), in which the value of y is calculated by substituting the value of x into the polynomial. When all group members combine their data points, the shared secret can be reconstructed by using a mathematical process called polynomial interpolation, which finds a unique polynomial going exactly through these points and extract its constant coefficient.

Our protocol begins when a group of $k+1$ users with $k+1 < n$, which is the total number of users of the organisation, wants to establish and share a common secret. Using the group size $k+1$ rather than k makes it simpler to describe all the mathematical formulae later. As mentioned above, this shared secret is the constant coefficient of a unique polynomial of degree k. To construct this polynomial, each user of the group will select a point with random x and y coordinates, so that the k-degree polynomial will go through the following $k+1$ points:

$$(x_0, y_0), \cdots, (x_j, y_j), \cdots, (x_k, y_k) \text{ with } 0 < j < k \tag{5.1}$$

To establish an interpolation polynomial in Lagrange form [Quadling, 1966], all x-coordinates

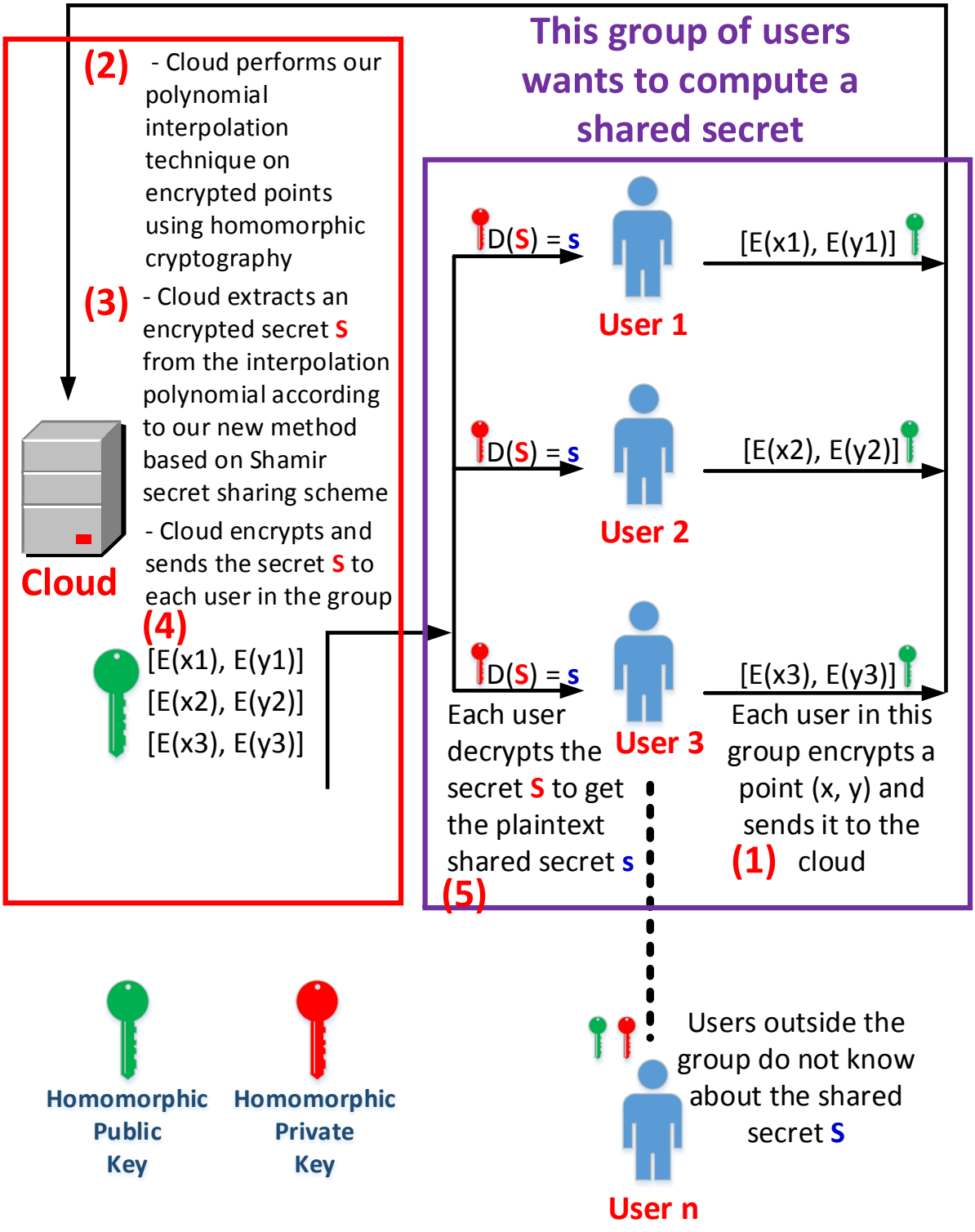


Figure 5.3: Computing a shared secret on the cloud for a group of users.

of the points above must be different, i.e. no two points having the same x-coordinate. This requirement must be met so that mathematical formulae in the intermediary steps described later do not contain any division by zero expression. This requirement is also reasonable because the interpolation polynomial is a function, which does not allow a one-to-many relationship. This requirements can be met in a number of ways using our model. For example, the $k+1$ users in the group can cooperate with each other in generating the unique x-coordinates for their points. There can be a group leader, who is responsible for managing the group and generating unique points for each member. The data-generating task can also be delegated to a trusted server residing inside the organisation.

After each of the $k+1$ users have got their points with unique x-coordinates, they will encrypt the x and y coordinates of their points using the homomorphic public keys as shown in Figure 5.3. These public keys are given to them using the key generation and distribution protocol described in the previous section and in Figure 5.2. The encryption function works be encrypting both x and y coordinates of each point. We assume that these coordinates have integer values. Each of these values will be converted to binary in the two complement format and each bit is encrypted using the selected homomorphic cryptography scheme proposed by Smart and Vercauteren [Smart and Vercauteren, 2010]. The encryption function $E()$ is applied to all the points as followed:

$$[E(x_0), E(y_0)], \cdots, [E(x_j), E(y_j)], \cdots, [E(x_k), E(y_k)] \text{ with } 0 < j < k \tag{5.2}$$

When a shared secret initialization request from the group and the encrypted points are sent to the cloud, it will perform the following major computing tasks:

- Executing the Lagrange polynomial interpolation process on encrypted points using homomorphic cryptography.
- Extracting the encrypted constant coefficient from the Lagrange interpolating polynomial using our new method based on Shamir secret sharing scheme.
- Sending copies of the ciphertext corresponding to the encrypted constant coefficient obtained from the above step to each user in the group. Users will use their homomorphic private keys to decrypt these ciphertexts and get the shared secret.

The first important computing task that the cloud performs on encrypted points is finding a unique polynomial going through all these points. Because all the x and y coordinates are encrypted, the cloud can compute on these data without knowing the actual values of x and

y, thereby protecting the security of all the points that we regard as shares of the secret. The output of this computing process is an encrypted polynomial in which only its order is known to the cloud while all of its coefficients are encrypted. However, when the data is encrypted, the possible computing operations that the cloud can perform on the encrypted points are limited. Specifically, only addition and multiplication over encrypted bits of the integer values of those points are permitted by the homomorphic cryptography scheme that we use.

In the plaintext domain, the unique polynomial L interpolating the set of $k+1$ data points, i.e. from point $P_0 = (x_0, y_0)$ to point $P_k = (x_k, y_k)$ with $k \in (0, n)$, can be written in a special format called the Lagrange form [Quadling, 1966] as followed:

$$L(x) := \sum_{j=0}^{k} y_j \ell_j(x) \tag{5.3}$$

In the above equation, y_j is the y-coordinate of a point P_j in the set of $k+1$ points, while $\ell_j(x)$ is a special polynomial called Lagrange basis polynomial which we will describe shortly. The above equation shows that the interpolation polynomial $L(x)$ is constructed by summing up all the scaled basis polynomials. Specifically, for each point P_j with $j \in [0, k]$, a basis polynomial $\ell_j(x)$ is created and then multiply with a scaling factor, which is the y-coordinate of P_j. However, when coordinates of the points are encrypted, the Lagrange form of the interpolation polynomial cannot be identified directly. Instead, we define an encrypted form of that polynomial in the following equation:

$$E[L(x)] := \sum_{j=0}^{k} E[y_j] E[\ell_j(x)] \tag{5.4}$$

where $E[y_j]$ is a homomorphic encryption of the y_j coordinate of a point P_j in the set of $k+1$ points and $E[\ell_j(x)]$ is a homomorphic encryption of the Lagrange basis polynomial which we will define later.

In the plaintext domain, basis polynomials $\ell_j(x)$ with $j \in [0, k]$ are main building blocks of the interpolation polynomial $L(x)$. For each point P_j, a Lagrange basis polynomial is constructed using the following equation:

$$\ell_j(x) := \prod_{\substack{0 \le m \le k \\ m \ne j}} \frac{x - x_m}{x_j - x_m} = \frac{(x - x_0)}{(x_j - x_0)} \times \cdots \times \frac{(x - x_{j-1})}{(x_j - x_{j-1})} \times \frac{(x - x_{j+1})}{(x_j - x_{j+1})} \times \cdots \times \frac{(x - x_k)}{(x_j - x_k)} \tag{5.5}$$

In our model, for each encrypted point $[E(x), E(y)]$, we define an encrypted basis polynomial

$E(\ell_j(x))$ based on the definition of Lagrange polynomial as shown below:

$$E(\ell_j(x)) := \prod_{\substack{0\leq m\leq k \\ m\neq j}} \frac{x - E(x_m)}{E(x_j) - E(x_m)} = \frac{(x - E(x_0))}{(E(x_j) - E(x_0))} \times \cdots \times \frac{(x - E(x_k))}{(E(x_j) - E(x_k))} \tag{5.6}$$

The encrypted polynomial above is constructed by the cloud for each encrypted point $E(P_j)$ with $j \in [0, k]$. The cloud only knows about the degree k of this polynomial and the indices of the encrypted points it should use.

Our goal is to compute the constant coefficient of the interpolation polynomial $L(x)$. However, this computing task must be performed as much as possible on the cloud using the given encrypted points to maximise data security and privacy. Therefore, we present here our computing steps that the cloud can execute using only the public key and the possible homomorphic computing operations allowed by the cryptography scheme. First, we need to compute the constant coefficient of each encrypted basis polynomial $E(\ell_j(x))$. From equation 5.6, the encrypted constant coefficient of $E(\ell_j(x))$ can be calculated in the following equation:

$$E(\text{Const\_Coeff}_{\ell_j(x)}) = \prod_{\substack{0\leq m\leq k \\ m\neq j}} \frac{-E(x_m)}{E(x_j) - E(x_m)} = \prod_{\substack{0\leq m\leq k \\ m\neq j}} \frac{E(x_m)}{E(x_m) - E(x_j)} \tag{5.7}$$

The equation above together with the definition of the encrypted interpolation polynomial $E[L(x)]$ shown in equation 5.4 allows us to deduce a formula to calculate the encrypted constant coefficient of $E[L(x)]$ as shown below:

$$E(\text{Const\_Coeff}_{L(x)}) = \sum_{j=0}^{k} E(y_j) E(\text{Const\_Coeff}_{\ell_j(x)}) \tag{5.8}$$

$$= \sum_{j=0}^{k} \left[E(y_j) \prod_{\substack{0\leq m\leq k \\ m\neq j}} \frac{E(x_m)}{E(x_m) - E(x_j)} \right]$$

$$= \frac{E(y_0)E(x_1)\ldots E(x_k)}{[E(x_1) - E(x_0)] \times \ldots \times [E(x_k) - E(x_0)]} + \ldots$$

$$+ \frac{E(y_k)E(x_1)\ldots E(x_{k-1})}{[E(x_0) - E(x_k)] \times \ldots \times [E(x_{k-1}) - E(x_k)]}$$

Equation 5.8 shows that computing $E(\text{Const\_Coeff}_{L(x)})$ requires a division operation. However, the homomorphic cryptography scheme in our model does not allow a division operation to be computed on ciphertext. Therefore, we will keep $E(\text{Const\_Coeff}_{L(x)})$ as a fraction having its

numerator and denominator as in the following equations:

$$E(\text{Const\_Coeff}_{L(x)}) = \sum_{j=0}^{k} \frac{N_j}{D_j} \quad \text{in which} \tag{5.9}$$

$$N_j = E(y_j) \prod_{\substack{0 \le m \le k \\ m \ne j}} E(x_m) \qquad \text{and} \qquad D_j = \prod_{\substack{0 \le m \le k \\ m \ne j}} [E(x_m) - E(x_j)]$$

The equations above show that expressions such as $N_0, N_1, \cdots, N_k$ and $D_0, D_1, \cdots, D_k$ contain only addiction, subtraction and multiplication operations. Therefore, they can be computed directly on the cloud using homomorphic cryptography to produce encrypted results.

Next, we try to compute $E(\text{Const\_Coeff}_{L(x)})$ further, by expanding the sum given in sigma notation in equation 5.9 into explicit sums using expressions in the forms of N_j and D_j as building blocks. This process is described in the following equation:

$$E(\text{Const\_Coeff}_{L(x)}) = \sum_{j=0}^{k} \frac{N_j}{D_j} = \frac{N_0}{D_0} + \frac{N_1}{D_1} + \ldots + \frac{N_{k-1}}{D_{k-1}} + \frac{N_k}{D_k} \tag{5.10}$$

$$= \frac{N_0 D_1 \ldots D_{k-1} D_k + \ldots + N_k D_1 \ldots D_{k-2} D_{k-1}}{D_0 D_1 \ldots D_{k-1} D_k}$$

$$= \frac{\sum_{j=0}^{k} \left(N_j \prod_{\substack{0 \le m \le k \\ m \ne j}} D_m \right)}{\prod_{n=0}^{k} D_n} = \frac{E(S_n)}{E(S_d)}$$

The cloud will compute both the numerator and denominator of $E(\text{Const\_Coeff}_{L(x)})$ in the encrypted domain using homomorphic computation as described in the equation above. The results of this process are two ciphertexts, $E(S_n)$ and $E(S_d)$, corresponding to the encrypted numerator and denominator, respectively. The cloud will then send copies of both $E(S_n)$ and $E(S_d)$ to each user in the group of $k+1$ members as shown in Figure 5.3. The figures also show how each user in the group will use his or her private homomorphic key to decrypt these ciphertexts, i.e. $S_n = D(E(S_n))$ and $S_d = D(E(S_d))$ and perform a division operation to obtain the shared secret $S = S_n \div S_d$. From one of our assumptions, which have already been mentioned at the beginning of this interpolation process, if each user in the group sends a point with a unique x-coordinate, the constant coefficient of the interpolation polynomial (also known as the shared secret S in our model) will be unique. Furthermore, equations 5.8, 5.9 and 5.10 show that the denominator of the constant coefficient, which depends only on the x-coordinates, is also unique if the x-coordinates are unique. Therefore, the cloud can send only the numerator $E(S_n)$ to the users to decrypt and keep as a shared secret.

5.3.2 Cloud-based Homomorphic Access Control Using Encrypted Secret Sharing

In this section, we propose an application of our encrypted secret sharing scheme described previously. This application is an access control model that allows a group of users to manage access to encrypted documents stored on a semi-trusted cloud. Our model has two stages. The first stage is about encrypting the document before storage on the cloud and sharing the access control right among members of a group. We will describe how a document is created, classified and encrypted before storage and how a group of managers can have equal access control rights using our proposed homomorphic secret sharing approach. In the second stage, we show how a document stored on the cloud with a classification code can be accessed only with the authorisation of all members of the group managing access. In these two stages, we will demonstrate how our homomorphic-based secret sharing scheme has played an essential role in protecting shares of the access key as well as enhancing the security of documents stored on the cloud.

The first stage begins when an author has created a document **D** and wants to store **D** on the cloud with restricted access as shown in Figure 5.4. We assume that the cloud is semi-honest, which means that it will follow the protocol, but may record all data collected during the execution of our protocol. Therefore, it is very important that all data stored and processed on the cloud is encrypted to ensure data security and user privacy. This requirement is addressed in our model by using a trusted server to encrypt document **D** with a symmetric key **S** before sending **E(D)**, the encrypted vesion of **D**, to the cloud for storage. This process is also illustrated in Figure 5.4.

In our model, restricting access to document **D** means that the author of **D** will classify this document into one of many different access control categories already specified by the organisation [Ferraiolo et al., 2003]. Assuming that document **D** is classified with a classification code **C** which is managed by a group of n manager, $M_1, M_2, \cdots, M_n$. Any user who wants to access documents in this category must ask for permissions from all members of this group. If one manager does not allow the user the access right, the document will not be accessible by that user even if all other managers have authorised access. We address this problem by an approach inspired by Shamir's secret sharing scheme [Shamir, 1979]. The symmetric key used to encrypt and decrypt document **D** is considered in our model as the secret **S** that needs to be shared among n managers. To achieve this, the trusted server will first construct a polynomial

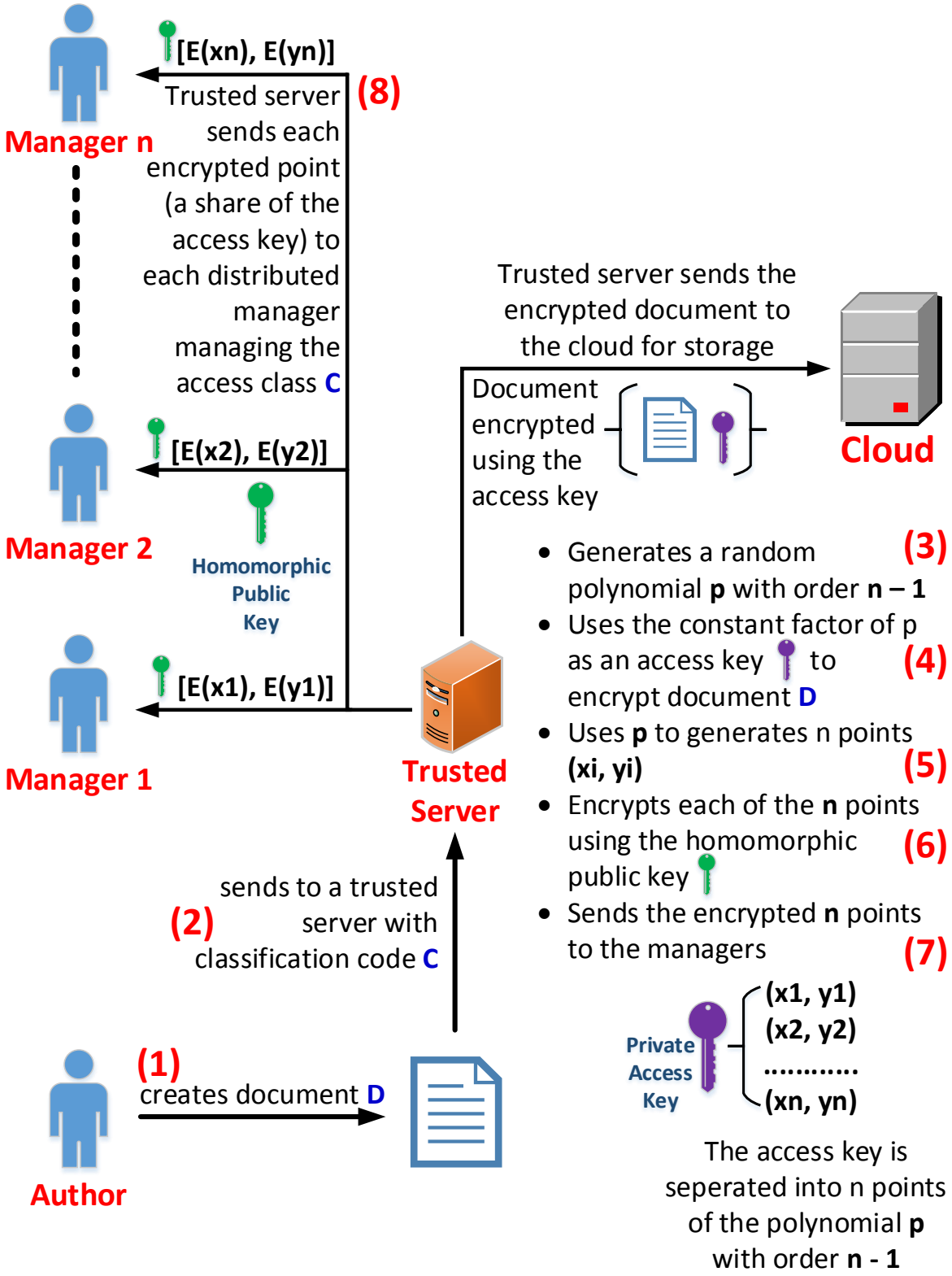


Figure 5.4: A document is encrypted by a symmetric access key and stored on the cloud while shares of the access key are distributed among members of the group controlling access to the document.

$\mathbf{P}(\mathbf{x})$ of degree $n-1$ as required by Shamir's scheme in the following equations:

$$P(x) = \sum_{i=0}^{n-1} a_i x^i = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots + a_2 x^2 + a_1 x + a_o$$

$$\text{with } a_0 = S$$

The polynomial $\mathbf{P}(\mathbf{x})$ must be random which means that all of its coefficients a_{n-1}, a_{n-2}, $\cdots$, a_2, a_1, a_0 must be chosen at random by the trusted server. The secret $\mathbf{S}$ is the constant coefficient $\mathbf{a_0}$ of $\mathbf{P}(\mathbf{x})$. The trusted server then construct n points (x, y) by randomly selecting n values for the x-coordinates and computing the corresponding y-coordinates using $\mathbf{P}(\mathbf{x})$. A requirement when selecting x-coordinates is that their values must be unique. Mathematical results in Shamir's secret sharing scheme [Shamir, 1979] and Lagrange interpolation polynomial [Quadling, 1966] have shown that given n points $\{P_{n-1},\ P_{n-2},\ \cdots,\ P_2,\ P_1,\ P_0\}$ with unique x-coordinates, it it possible to construct a unique polynomial of degree $n-1$ interpolating those points. Therefore, after using the constant coefficient $\mathbf{a_0}$ of $\mathbf{P}(\mathbf{x})$ as a secret symmetric key $\mathbf{S}$ to encrypt the document, the trusted server can dispose $\mathbf{P}(\mathbf{x})$ because the n points $\{P_{n-1},\ \cdots,\ P_0\}$ are sufficient to re-generate $\mathbf{P}(\mathbf{x})$ and obtain $\mathbf{a_0} = \mathbf{S}$ at later stages. Each point in the set $\{P_{n-1},\ \cdots,\ P_0\}$ is considered a share of the secret $\mathbf{S}$ according to Shamir's secret sharing scheme [Shamir, 1979]. The trusted server will encrypt each point and send it to the corresponding manager in the group managing access to the classification category $\mathbf{C}$ as shown in Figure 5.4. At the end of this stage, the trusted server uploads the document encrypted by the secret key $\mathbf{S}$ to the cloud while each manager will possess a share of that secret key.

The second stage of our access control model is when a user in the organisation wants to access a file stored on the cloud. At this stage, the file is already encrypted by a symmetric access key $\mathbf{S}$ and stored on the cloud while each manager is keeping a share of $\mathbf{S}$. Getting access and retrieving the content of this file require the user to ask for permissions from all managers in the group. An access request sent from the user to the managers can include information such as identification details of the user and the file, purposes of the access and any other information that are required to approve the access. Upon receiving an access request from a user, each manager has the right to approve or disapprove this request. Because of the way we have constructed and distributed shares of the secret $\mathbf{S}$ at the initial stage of our model as shown in Figure 5.4, the symmetric access key $\mathbf{S}$ can only be recreated when all of its shares are combined. Therefore, if one share is missing, the document cannot be decrypted.

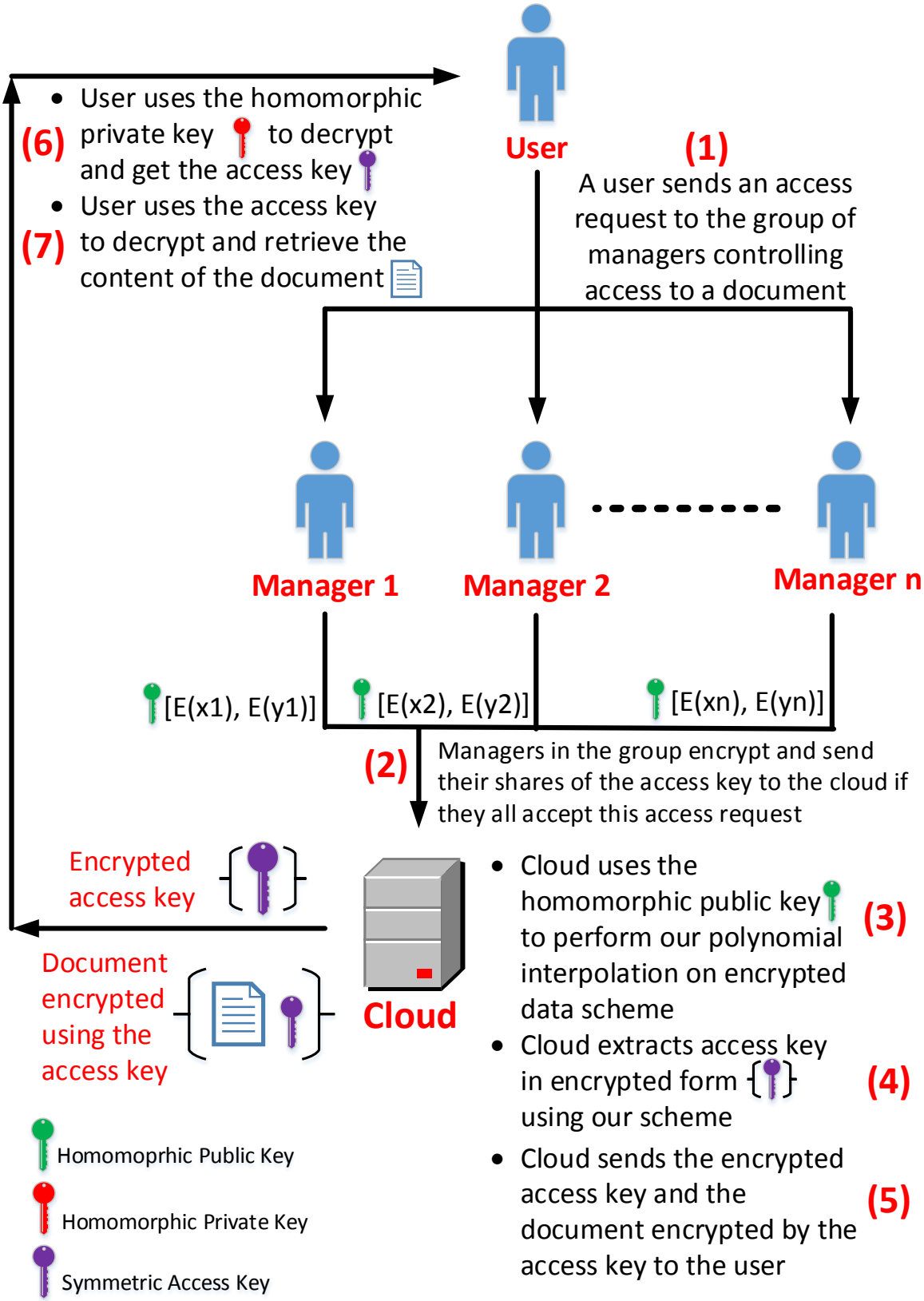


Figure 5.5: A user who wants to access an encrypted document stored on the cloud needs to have permission from all managers in the group controlling access to that document.

When all managers have approved an access request, they will have to combine their shares in order to retrieve the secret **S**. Our model allows the combination of shares and the computing operations to retrieve the shared secret to be performed on a semi-trusted cloud as shown in Figure 5.5 without disclosing a manager's share of the secret to the cloud or other managers. To achieve this, each manager will encrypt his or her share of the secret **S** using the homomorphic public keys distributed to them earlier as shown in Figure 5.2. Each share of the secret is an encrypted point **P** having its x and y-coordinates encrypted separately.

After receiving an access request to a document from a user, the cloud will check if all managers have authorised access based on the number of managers who have sent their encrypted shares of the secret. If at least one manager doesn't response, the cloud informs the user that he or she does not have access to that document. However, if the cloud receives encrypted points from all the managers in the group, it will start a homomorphic computing process to recreate the encrypted secret key **S** directly on those encrypted points using only the public homomorphic key as shown in Figure 5.5. This is where we apply our homomorphic encrypted secret sharing approach proposed earlier. Homomorphic computation allows the interpolation polynomial of degree $n-1$ to be constructed on encrypted domain in the following format:

$$\sum_{i=0}^{n-1} E(a_i)x^i = E(a_{n-1})x^{n-1} + E(a_{n-2})x^{n-2} + \ldots + E(a_2)x^2 + E(a_1)x + E(a_o)$$

$$\text{with } E(a_0) = E(S)$$

The encrypted polynomial above is unique and interpolates all n encrypted points $\{E(P_{n-1}),$ $E(P_{n-2}),$ $\cdots,$ $E(P_2),$ $E(P_1),$ $E(P_0)\}$ sent from the managers. Furthermore, all coefficients are encrypted which means that the cloud does not compromise data security during the homomorphic computation process.

One advantage of using our approach is that the cloud does not need to calculate all the encrypted coefficients of the interpolation polynomial. Only the encrypted constant coefficient $E(a_0)$ needs to be computed because it has been used as the symmetric access key to encrypted the document as shown in the first stage of our access control model. Therefore, $E(a_0) = E(S)$. The cloud will compute both the numerator and denominator of $E(a_0)$ in encrypted domain as described in Equation 5.10 in the previous section. Finally, the cloud sends both the encrypted document and the encrypted access key, comprising of the encrypted numerator and denominator of the constant coefficient, to the user requesting access. Because the access key is encrypted by the public homomorphic key, if the user belongs to the organisation and possesses a private homomorphic key according to our key distribution model shown in Figure 5.2, he or

she will be able to perform a decryption. Furthermore, if all managers have provided the right encrypted shares of the access key, then the decryption process will output the correct access key which the user can use to decrypt and retrieve the content of the document. However, one missing or incorrect encrypted share is sufficient to cause a false share secret to be computed and decrypted, making it impossible to decrypt and retrieve the right content of the document.

5.4 The Security of Our Model in Various Scenarios

In this research, we have introduced a new approach to sharing a secret using homomorphic cryptography which can be very useful in the management of cryptographic keys. While cryptography schemes such as symmetric and asymmetric encryptions can be employed to protect the integrity and security of data, securing and distributing the keys used in such schemes are also important problems that need to be addressed thoroughly. However, using further encryptions to protect the keys just change the key protection problem rather than solving it by shifting the focus of security from one key to another. By dividing a secret key into multiple shares, our scheme differs significantly from traditional approaches such as keeping the key in a single, well-protected location or storing several copies of the key at different locations.

The feature that makes our scheme similar to other secret sharing schemes is the fact that a shared secret is separated into multiple pieces and the requirement that multiple share holders need to combine their shares in order to regenerate a secret. However, our scheme will not work when one share holder does not authorise the key reconstruction by withholding his or her share of the secret. This is different from other threshold secret sharing schemes such as Shamir's secret sharing [Shamir, 1979] in which a secret can be recovered when k out of n shares are combined. Another issue is securing all shares in the process of regenerating the secret. Some secret sharing schemes require all the shares to be combined in plaintext so that computing operations can be performed directly over the shares to produce the final result. This approach can be fast and efficient. However, its limitation is that share holders cannot protect their shares of the secret from the party which performs the computing operations. Our scheme overcomes this limitation by using homomorphic cryptography to encrypt all the shares before the secret reconstruction process begins. Therefore, share holders can send their encrypted shares of the secret to a semi-trusted cloud for reconstruction work without compromising data security.

As a result of these similarities and differences, choosing an appropriate secret sharing

scheme will depend significantly on the application scenario. Sharing a secret by distributing multiple copies is convenient but easy to misuse. Using a threshold (k, n) secret sharing scheme like the work of Shamir is suitable in cases where a sufficiently large majority can authorise an action, but a high level of trust should be placed on the party collecting and combining all the shares. Our scheme can guarantee that the shares are protected by means of encryption, especially when most of the computing operations required to reconstruct the secret are performed on encrypted data. Our scheme is also safe and robust in that it requires the participation of all share holders. However, it may be inconvenient in some applications because even if one member does not authorise his or her share, the secret cannot be reconstructed.

The security of our model also relies on the security of the cryptographic technique that we use, which is the homomorphic scheme proposed by Smart et al. [Smart and Vercauteren, 2010]. In their work, the authors have proven three security features of their scheme, i.e. key recovery, onewayness of the encryption and semantic security. In many homomorphic computing operations, the cloud often use the public key extensively to compute over encrypted data, re-encrypt the ciphertexts and output encrypted results. However, the construction of the scheme makes it very difficult to find an efficient solution to retrieve the private key given only the public key by relying on the hardness of the Small Principal Ideal Problem [Smart and Vercauteren, 2010] as the mathematical proof of its difficulty. The onewayness of encryption is the ability to recover a plaintext message given its corresponding ciphertext. In their security proof, the authors compare this problem to the closest vector problem [Hoffstein et al., 2008], which is also a hard problem in lattice theory. The scheme is also proven to have semantic security which means that different ciphertexts are produced when the same plaintext is encrypted multiple times, making it impossible for another party to replay the encrypted information obtained during transmission.

Next, we analyse the behaviours of our models under different use case scenarios. These can be typical cases in which the security and integrity of the system are not affected or can be the cases where a malicious user tries all the possible methods to attack the system. Targets of such attacks are the shares of the secret or the encrypted document protected by the shared secret. Through these cases, we show that our model has all the essential features to deal with such attacks and protect the shares of the secret as well as encrypted documents stored on cloud.

- Scenario 1: One or more managers in the managing group do not authorise the access request.

Our model is designed so that the access of an encrypted document stored on the cloud is controlled by a group of users. If one user in the group does not authorise access, the document cannot be decrypted and the content cannot be retrieved. To enforce this rule, the cloud will keep a list of all members controlling access to a particular document or a document class. Once an access request is received, the cloud will send a notification to each member of the group asking for authorisation. Each group member will approve this request by sending his or her encrypted share of the secret. If the cloud receives the number of shares equal to the number of group members, it knows that access has been authorised and continues on to perform computation on encrypted shares and retrieves the encrypted access key. On the other hand, if the number of encrypted shares received by the cloud is smaller than the number of members controlling access to the document, the cloud will know that the access request is not authorised and will send a notification to the user requesting access.

- Scenario 2: An unauthorised user tries to get access to a shared secret.

  In our model, the access key to an encrypted document or a document class is divided into multiple shares by a trusted server. Each share is then distributed to a member of the group controlling access. However, if the shares of the access key are stored in plaintext, it will not be secure in cases when there are deliberate attempts to obtained unauthorised access the system. Therefore, shares of the secret stored on computers of managers need to be encrypted by a unique password or pass phrase known only to that manager. To achieve this security requirement, we propose the use of a combination of Triple DES, which is a symmetric cryptographic scheme and the hash function SHA-256. After receiving a share of the secret, the manager will select his or her own pass phrase and pass it through the hash function and use the output of the hash function as a symmetric key for the Triple DES scheme to encrypt his or her share of the secret. This method allows shares of the secret to store in encrypted form on managers' computers. Therefore, even if the security of their computers have been breached, unauthorised users cannot access the shares in plaintext because they do not know the pass phrase.

- Scenario 3: One manager in the control group lost his or her share of the secret.

  From the previous use case, it is guaranteed that each manager has a share of the secret

stored in encrypted format on his or her computers, thereby protecting the security of the share. However, the integrity of the shares can be violated because it can be accidentally or deliberately deleted by the managers or malicious users obtaining unauthorised access to the encrypted shares. Therefore, share owners can take precautionary methods to protect the integrity and availability of their shares by making multiple copies of their encrypted shares and store these copies separately. In the worst case scenario when a share owner cannot present his or her share, the whole secret sharing and key generation process will be computed again by a trusted server. However, the possibility of such computation is minimised because of the precautionary method described above.

- Scenario 4: A man-in-the-middle attack happens in which a malicious user intercepts the encrypted shares of the secret and replays them later.

  When approving an access request, each manager needs to send his or her encrypted share of the secret to the cloud. This stage is susceptible to man-in-the-middle attacks in which a malicious hacker can intercept the data and replay the encrypted shares later. To prevent such attacks, our scheme can allow the managers to generate and send one-time encrypted shares of the secret to the cloud. Each request access sent to the manager for approval can contain a unique pin code for that access. Each manager can encrypt both the code and the shares of the secret using his or her private key and then run a hash function to generate a unique signature for that access, making it much more difficult for replaying such results in the future.

5.5 Implementation and Experiments

To implement our model, we have built a software prototype using the C programming language to simulate all the steps described in our model. Because this cryptographic scheme is used to encrypt binary data, we assume that all inputs are discrete numbers that can be converted to integers. Encrypting an input means that each bit of its binary representation is encrypted using the public homomorphic key. Likewise, decrypting an output of a homomorphic function requires each ciphertext to be decrypted separately using the homomorphic private key to obtain an array of binary digits, which will subsequently be converted to an integer. In our implementation, we wrote several software modules to convert integers into their corresponding two complement binary representation and back from binary to integers ready for encryption

and decryption processes, respectively.

The key generation, encryption and decryption was implemented using the Scarab homomorphic cryptography library [Perl, 2011]. It was written in the C language as an implementation of the Smart and Vercauteren [Smart and Vercauteren, 2010] homomorphic encryption scheme. This open source library was created to provide the developer community and researchers with a platform to build further applications and experiments using homomorphic cryptography. It allows us to focus on building the main functionalities of our model rather than the low level implementation details. In many of our experiments with homomorphic encryption, we needed to experiment with large integers, for example, when creating large prime numbers or when storing results of several multiplication involving large integers. However, the built-in integer type of the C programming language, which has a size limit of 4 bytes on a 32 bits system or 8 bytes on 64 bits system, is not large enough for our experiments. Therefore, we used the GNU Multiple Precision Arithmetic Library [GNU, 2014] (GMP), which is an open source and portable library written in C for arbitrary precision arithmetic on integers. GMP is designed to give good performance for our applications in cases when we needed a few hundred bits of precision by carefully keeping the overhead at a minimum. Other C libraries were also used to handle store and process large integers such as the The Multiple Precision Floating-Point Reliable Library [Guillaume Hanrot, 2013] and the Fast Library for Number Theory [Hart, 2013].

Our model and these libraries were run on a machine having 4 Gigabytes of memory and 2.4 Gigahertz quad core processor with Ubuntu Linux as the operating system. We represented each large integer in memory by a large vector of bits. The encryption process receives an input bit and outputs a ciphertext which is a large integer. Therefore, an encrypted integer is represented as an array of large integers corresponding to each bit of the two complement binary representation of that integer. To evaluate the performance of our software modules, we write codes to measure the time each task were executed directly in every functions.

In the first experiment, we began with measuring the time taken to generate the public and private homomorphic key pair. This process requires the generation of large prime numbers. A few large random numbers were generated and their primality were checked until all primality tests were satisfied. Therefore, we obtained a small variation in the execution time of the key generation algorithm. Next, we measured the time taken to encrypt and decrypt one bit using the homomorphic key pair that has been generated. Finally, we measured the time taken to execute homomorphic addition and multiplication given two ciphertexts and the public homomorphic key. Table 5.1 shows the results of all these experiments described above.

Next, we performed experiments to construct and shared a secret to many groups having different group sizes. This is the central part of the implementation process because it requires all the core functionalities to be implemented and executed. The built-in integer type of the C language was not enough to store the intermediate and final results of many multiplication operations, therefore we selected the integral type provided by the GNU Multiple Precision Arithmetic Library. Ths type allows arbitrary precision arithmetic computing operations to be performed and uses as much memory as possible to store the results.

In the first stage, we wanted to test all the functionalities and correctness of our model using arbitrary precision arithmetic. There were many factors and adjustments to consider such as the group sizes, i.e. the number of members with whom a secret can be shared or the magnitude of the coefficients of the interpolation polynomial. These parameters will significantly affect the size of the intermediate results which have a direct effect over the size of each binary number we chose in the encrypted computing process. With arbitrary precision arithmetic library, our model worked as it was expected. The secret is the constant coefficient of a polynomial with other coefficients chosen at random. With a specified group size, the secret is shared among each group member, each of them keeps a tuple containing the x and y coordinates of the corresponding points of the interpolation polynomial.

In the second stage, instead of using arbitrary precision arithmetic, we performed experiments directly with encrypted binary numbers. We wrote our own software modules to convert decimal to binary numbers using the two complement format. Then each binary digit was encrypted and computing operations were performed directly on the corresponding ciphertexts. With this approach, the binary word length is also an important parameters deciding the correctness of the computation. A short word length, for example 4 bits, was not enough to store results of the computation. Therefore, we experimented with many word lengths, specifically, we used 8-bit and 16-bit word lengths. Table 5.2 shows the results and performance of our method with various word lengths and group sizes. Although our model is designed so that a secret can be shared among members of a group having arbitrary size, in reality, computing on encrypted binary has shown some technical limitations that need to be addressed.

The execution time of our model, as shown in Table 5.2, increases with larger group sizes, from around 1 minute to 20 minutes when a secret is shared with 2 to 5 members. This execution time may decrease significantly when our model is executed on cloud servers with larger computing capacity. However, with current homomorphic cryptography scheme, we can only ensure the accuracy of the results when small coefficients of the interpolation polynomial were used and when the group size is around 5. This is mainly because we are using a cryptography

scheme which still have large ciphertext size, causing the intermediate results to be overflown when using large group size. However, our models can be applied to any cryptography scheme with homomorphic addition and multiplication. Therefore, in the future, we can use our model with more efficient homomorphic schemes for larger group sizes.

Homomorphic Operation	**Time**
Key Generation	From 5 to 25 (seconds)
Encryption	12.24 (milliseconds)
Decryption	15.72 (milliseconds)
Addition (homoXOR)	3.28 (milliseconds)
Multiplication (homoAND)	5.19 (milliseconds)

Table 5.1: Execution time of homomorphic operations

Group Size	Encrypted Multiplication Count	Encrypted Addition Count	homoXOR Operation Count	homoAND Operation Count	Total Execution Time (mins)
2	5	3	10006 (8b) 40512 (16b)	6288 (8b) 25376 (16b)	1.07 2.16
3	12	7	24032 (8b) 97216 (16b)	15088 (8b) 60896 (16b)	2.56 5.18
4	27	13	53984 (8b) 218560 (16b)	33904 (8b) 136928 (16b)	5.76 11.64
5	49	23	97952 (8b) 396608 (16b)	61520 (8b) 248480 (16b)	10.45 21.12

Table 5.2: The execution time of our method using 8-bit and 16-bit binary word length when a secret is shared among groups having 2 to 5 members

5.6 Conclusion

In this research work, we have shown how homomorphic cryptography can be combined with secret sharing to create a useful application enabling a group of users to control access to encrypted documents stored on cloud server. Using homomorphic addition and multiplication capability of the encryption scheme, we show how most of the computing works can be performed on encrypted data, thereby preserving the privacy of users and the confidentiality of data stored and processed on the cloud. We have also evaluates the performance of our

scheme by conducting many experiments using arbitrary precision arithmetics and directly on encrypted binary digits. In future experiments, we can apply our model with more efficient homomorphic schemes to use with larger group sizes and better performance.

Chapter 6

Secure one-time e-commerce transactions using homomorphic cryptography

Abstract

In this final chapter of the thesis, we apply the comparison feature of homomorphic cryptography to secure sensitive data such as personal and financial information in an e-commerce transactions of an online business. Not only the data must be secured when stored on servers, but the integrity of these data must also be guaranteed in transit, so that any financial information which has already been used cannot be replayed in another transaction without the permissions of data owners. Recent incidents involving major data breaches in large e-commerce organisations have proven that such potential privacy threats can severely impact the rapid development of electronic commerce. Therefore, in this research, we propose a secure e-commerce transaction scheme with strong privacy preserving features which are built using homomorphic cryptography. We apply this novel cryptographic scheme to protect customer sensitive information from all the parties involved in an online transaction as well as ensuring the authenticity of business data related to that transaction in storage on any servers or in transfer across the Internet. We also implement a novel mechanism allowing all authentication details of a business transaction to be non-reusable immediately after they have been used for the first time. This important feature enables users of our model to purchase goods anonymously from an online merchant while guaranteeing the authentication and integrity of data.

6.1 Introduction

The number of business transactions executed over the Internet has increased significantly due to the growing number of online users making various selling and buying activities over high speed broadband or wireless network connections. Furthermore, the types of devices that are used for e-commerce activities can vary significantly from personal desktop computers, gaming consoles, smart television sets to portable devices such as notebooks or from various applications running on mobile devices. It is no longer sufficient for any online merchant to just manage the typical methods of selling goods and services. There are other alternatives to make online payments such as using various types of cards or even not using a card at all such as in payments by mail orders. These changing facets of e-commerce always require more effective and efficient mechanisms to protect data security and privacy of a large number of users participating in online transactions and electronic payments.

There are major differences when conducting online transactions compared to other more traditional commerce activities. After selecting an item at a supermarket, for example, a customer only needs to proceed to the counter and make a payment, typically by cash or card. Having a face-to-face meeting with customers means that merchants do not need to store a lot of details about their customers as long as the purchases items have already been paid for. On the contrary, when making online payments, there are a lot more activities going on behind the scenes through out the process of executing those transactions. The popularity and widespread nature of the Internet means that customers of an online business often do not meet merchants in person. Instead, they can make online purchases everywhere in the world using different types of devices and payment methods. Online merchants need more information about the customers to ensure that all the online transactions are genuine and payments will eventually be made legally to the right destinations. To achieve these purposes, customers need to provide details such as residential and delivery addresses , payment types, card details, delivery date, etc. Furthermore, merchants can implements online mechanisms to check the browsing patterns of customers visiting their websites, revealing as many details as possible to build a profile of those customers. The behaviours and interest of online customers are also valuable sources of information for e-commerce websites to show targeted advertisements or recommend relevant items relating to a particular purchase.

As more and more people are making online purchases at any time around the world, a greater than ever amount of personal and sensitive data is being collected by millions of e-commerce servers around the world. While these customer data are very important for

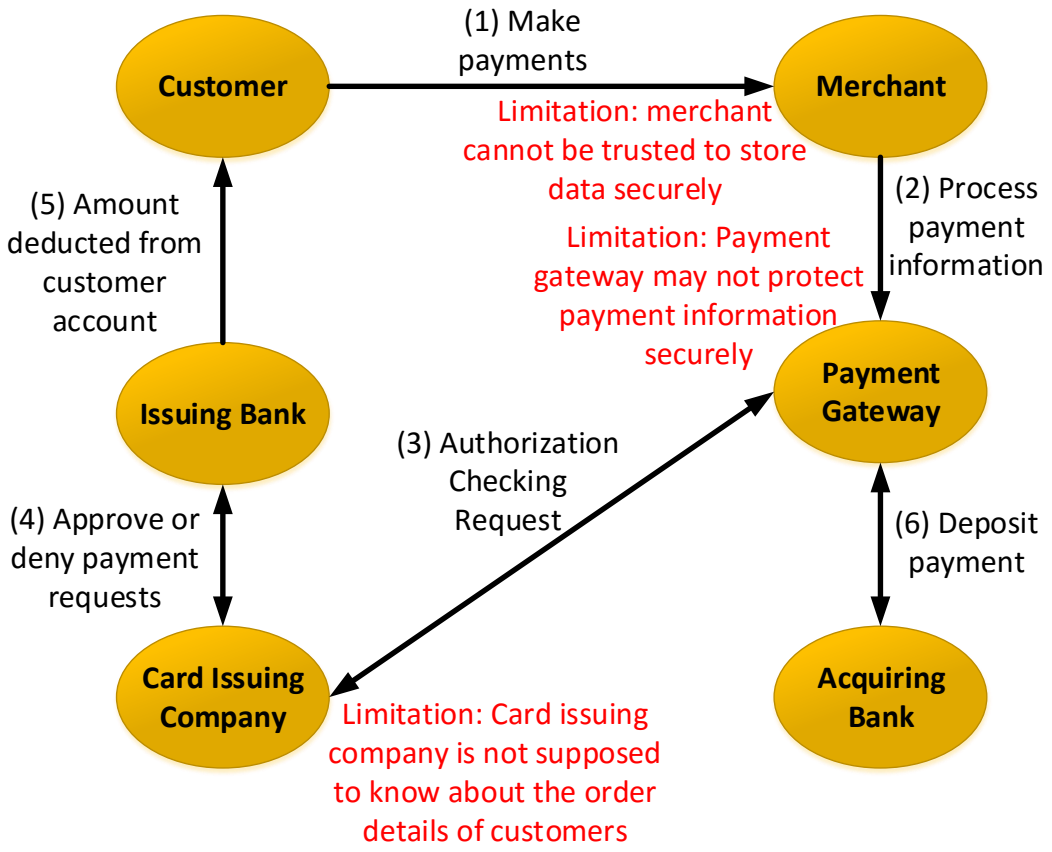


Figure 6.1: Limitations of existing online payment processing system.

verification purposes and studying user behaviours, the actual data owners, i.e the customers, often have little to no control over how the information flows from their devices to distant servers. Furthermore, data owners do not have the authority to specify how their sensitive payment information is collected and stored. When making an online purchase, customers often do not spend time to read the privacy policies regarding the management of personal identifiable data of a particular website, making it quite likely that customers might inadvertently reveal more personal details and sensitive information than necessary to online merchants. The data given by customers are usually stored in remote servers located anywhere in the world under the jurisdiction of different rules and regulations regarding data security, integrity and user privacy. These limitations are demonstrated in the overview diagram of online payment processing system as shown in Figure 6.1.

The aspects of online businesses described above are potentials for privacy threats which can be a huge setback for the development of electronic commerce, affecting both the customers,

the payment gateway companies, the online merchants and all other parties involved in online transactions. Many security breaches happening to customer data stored on remote servers have actually helped customers and users of electronic payment systems to be more aware of their online privacy and what happen to their payment information. Therefore, when customers have doubts about the security of a website, they simply do not make any purchase at all because of the hesitation of being involved in a card fraud or identity theft. From the perspective of e-commerce service providers, failing to provide adequate measures to protect customer data will eventually lead to a decreasing number of customers using the services they provide, causing significant loss of revenue.

Before proposing and implementing solutions to protect data security and privacy of users participating in e-commerce transactions, it is very necessary for us to provide clear descriptions of all the possible privacy concerns to the best of our knowledge. Firstly and most importantly is the privacy of information that is exchanged over the course of an online purchase, especially how to define the ownership and sharing of information. Privacy also includes the protection of confidential data during transit and all the frameworks and technical solutions provided by the online merchants to guarantee the solid confidentiality and integrity of their services. One of the most popular methods employed by merchant to address these privacy requirements is the use of Secure Socket Layer (SSL) protocol. The main purpose of this protocol is to provide data integrity and privacy between two computing application participating in a transaction over an Internet connection. Once an SSL connection is established, a client and a server will have a private and reliable connection with authenticated identities of the communicating parties. This is made possible by the use of symmetric cryptography to encrypt the data transmitted and public key cryptography to authenticate the identity of at least one of the parties.

Although Secure Socket Layer technology has been a very effective tool to protect the privacy of communication lines, new research works are still required to protect the security, integrity and privacy of customer information, especially when online customers are required to provide more sensitive data than ever regarding items purchased and customer behaviours. If online users are not equipped with effective tools to manage these data, merchant servers usually assume the rights to manage and store such data in any way and as long as they wish. It would mean that customers must have a certain degree of trust that merchants will keep their data secure while in reality, there are cases in which the dishonesty or incompetence of merchants are often the causes of major privacy breaches. The JP Morgan Chase, which is one of the biggest data breaches in the year 2014 can be used as an example to illustrate our point. This bank is one of the largest in the United States in terms of assets before its publicised a

massive data breach affecting 76 million households and 7 millions small businesses. Although it was reported that no unusual customer fraud had resulted from this breach, a large amount of value customer information had been compromised. Personal identifiable information such as customer names, addresses, phone numbers and email addresses were stolen by hackers. Although the bank had played down the possibility that more sensitive bank details such as account numbers, passwords, date of births and social security numbers were also stolen, the breach is still a major security incident affecting world-wide customers using web and mobile services of the bank.

Contributions. To minimise the possibility of such security breaches as well as reduce their consequences to a minimum, many online merchants and financial institutions often employ the services of third party payment gateways. However, the disadvantages of using PSP is that attacks to compromise data security can still be directed towards PSP rather than the e-commerce websites themselves. A payment gateway still has to deal with major security issues such as insider attack or how to trust the employees in protecting customer information of many online businesses using their services. There have been many major security breaches in which a few payment gateways had to take the main responsibility. The severe limitations of both the online merchants and third party payment service providers have been the causes of an ever increasing number of major incidents compromising the data security of customers participating in e-commerce transactions. We need more robust technical solutions to enable the customers themselves to take direct control to protect the security, integrity and privacy of their data from all other parties participating in online transactions. Therefore, in this research work, we apply the latest developments in homomorphic cryptography to create a novel online payment system which can address the requirements mentioned previously. Furthermore, our payment system allows anyone to make purchases of goods and services from online merchants without actually revealing their sensitive and personal details. Thus, a transaction in our system can be made by our new mechanism which can cause payment details sent to online merchant to expire after the first use. This feature is very effective in enhancing customer privacy, especially in cases when an online merchant or a payment service provider lost control of their customer data, no one else will be able to reuse those information for illegal purposes. The contributions of this research work can be summarised as followed:

- Our payment system enables customers to purchase items anonymously while still allows the merchant to collect enough information to determine whether to trust a customer. We use the minimal set of personal information that a customer participating in an online

transaction needs to provide to distant parties, including what they possess and what they know. Our model process those data securely so that while they can be used in one transaction effectively, replay attacks by a middle party attempting to reuse such data will not be successful. This feature of our model not only benefits the customers, but also alleviating the responsibilities of merchants in protecting sensitive customer data without incurring any additional computational cost.

- We apply homomorphic cryptography scheme in our model to maintain confidentiality by compartmentalize sensitive data into different encrypted packets which are only accessible to relevant parties in a transaction. Therefore, merchants cannot access payment information of customers while the card issuing company or the payment gateway can process payment information without knowing exactly what items customers are buying. No dishonest or malicious users can intercept the communication line and decrypt the encrypted packets at any given time as long as the private keys of the merchants and the card issuing companies are adequately protected.

- Our model is carefully design to ensure the non-reusability of encrypted sensitive customer data. This feature is achieved by integrating a unique time-stamp and random session key when executing encryption information. Therefore, every time customers make a request for payment, the ciphertexts of the payment details will be unique, making it impossible to use such ciphertexts more than once. The possibilities of many security risks such as a replay attack or an identity theft can be minimized due to the implementation of this feature in our model.

6.2 Background

6.2.1 Major components of an online payment system

Once a business or retailer has decided to go online to sell their goods and take electronic payments, they must apply for a merchant account with a bank which is called an Internet Merchant Account (IMA). The bank itself is called the acquiring bank which is responsible for managing existing funds of the online merchant, authorising transactions and transferring the money to another account inside or outside this acquiring bank. As an alternative to using acquiring banks, online merchants can also use dedicated merchant account providers of various types of financial service companies. If merchants want to take the card-not-present payments

such as mail orders or using electronic funds transfer at point of sale (EFTPOS), they would still have to speak with a financial institution of their choice in order to acquire an Internet merchant account identification number for subsequent verification purposes. Every time the acquiring bank processes or receives a payment, it might take a small amount of money called the acquirer fee from the total sum of the transaction before depositing the money to the merchant account. Therefore, merchants should find out about any fee prior to signing the merchant service agreement with the acquiring bank and payment service provider. Another key component is any e-payment system is the issuing bank of customers. This type of bank issues credit cards such as Visa and MasterCard and is also called the card holder's bank. Today, there are many more types of cards that can also be used for online payments. One example is the regular debit card issued by the local bank of a customer. Large banks will cooperate with well known card issuing company to provide customers with cards that can use for both ATM and online transactions. There is a fee to be charged by the issuing banks which is called the interchange fee.

After creating an Internet Merchant Account, a merchant will have the option to use the services of a payment service provider (PSP). These payment service providers or payment gateways are often experienced agents with expert knowledge about the security of payment systems. They will function as a cashier or point-of-sale terminal to collect card details, check for fraudulent transactions and securely transfer all the details to the acquiring bank for processing. The advantages of using payment service providers are that they offer different rates and packages to suit the scale and requirements of different merchants with options to host the secure payment pages on the PSP servers or merchant servers. It very easy to a customer to see a PSP in action on the web pages where customers submit their payment details. In reality, some Internet merchant accounts are provided by payment service provider companies and some acquiring banks can also provide PSP services. Furthermore, if customers do not want to use the services of a PSP or IMA, they can select a payment processing company instead. With some advantages such as simplified application procedures and faster to set up, this choice is ideal for online merchants which have little or no trading history. However, many processing companies require customers to be sent to the websites of such companies in order to make payments. Other disadvantages are longer settlement time and higher cost than using PSP or IMA. Some typical examples of payment processing companies are PayPal and Google Checkout which might take significantly less time to process payments and deposit money to merchant accounts.

6.2.2 Online payment processing

Processing an online payment is a complicated procedure in involving many parties such as the customers with their cards, the issuing bank, the acquiring bank, and the merchants together with the combination of many e-commerce and web technologies. In this section, we explore step-by-step examples of a common credit card payment process, investigating how all the technologies and major components of an online payment system work together in two cases, i.e. when a card is used to purchase goods online and how a card-not-present payment can be made.

An online payment begins when a customer has added a product to the shopping basket and is ready to proceed to checkout. To enhance the confidence of customers, the basket page is often encrypted with Secure Socket Layer with a valid public key certificate granted by a well-know and trusted Certificate Authority (CA). This is a very important security feature because if the certificate expired or customers had doubts about the origin of the certificate and the CA, they could choose to cancel the transaction all together to protect their privacy and credit card information. When a customer proceeds to the checkout page, merchants will collect a lot of information from that potential customer. They may ask customers to create an account and log in so that all information relating to that customer will be retained for future purposes. The types of information merchants often collect at this stage is user name, full name, address, phone numbers, billing and shipping addresses. After this information is submitted, the merchant web services will validate all details send everything to the back-end database. If a payment service provider is used, a copy of this information will also be sent to the PSP and the waiting customers will also be directed there. For each order, the PSP also keeps the merchant ID, a unique order reference, the total amount of money to be charged for that transaction, in addition to all of the customer information mentioned above. After receiving and processing all the information, the PSP will return a secure key in a receipt verifying that particular set of data against the transaction.

Behind the scene, the PSP will authenticate all card details submitted by the customer. Next, the card holder total charge must be authorized and the funds allocated to the merchant. The payment can be deferred until the goods have actually shipped. The PSP also sends the final information about the transaction to the merchant, with details such as the unique order reference, security-related confirmation number and postal code checks. To keep a permanent storage of this information, merchants are often required to comply with industrial standards such as the Payment Card Industry Data Security Standard. This is a set of criteria established

by the authority to ensure a security environment when processing, storing or transmitting credit card information. Such standards are usually required by law to apply to all organisations and merchants at any sizes or scales, with any number of transactions in which card holder data is accepted, transmitted and stored. Merchants will risk the potential for severe daily fines of they do not comply with such standards.

6.2.3 Limitations of existing online payment models

Although there are many online payment processing models providing customer with flexible options to pay for goods and services from the Internet, there are still major limitations preventing such models from protecting data security and customer privacy more effectively. These limitations can come from both technical factors and specific design issues of such protocols. This section will describe all such limitations from the customer privacy perspective.

To put this discussion into the right context, it is essential to understand customer privacy at various levels from absolute security to partial security to no security at all. The highest security level is called absolute security. In this case, customers can purchase goods and services with minimal information sent to merchant servers. They can even make an anonymous purchase, meaning that the information stored on distant servers is not sufficient to determine identities of customers. Customers also have direct control of how their data is stored, preferably by means of cryptography so that even if servers are compromised, the damage to data security and privacy is kept to the minimum due to the robustness of the encryption scheme being used. An example of this type of privacy is the one-time security token to identify one transaction from another. This token can be established in many ways, for example, being sent to customer mobile phones or via another communication channel completely separated from where the transaction is happening. After its first use in a transaction, this one-time security token will become invalid and will be replaced by a new token when another transaction begins. Therefore, any dishonest individual or malicious party can not reuse this security token in any fraudulent transaction. Another level of security is partial security, which has the participation of a payment gateway. In many e-commerce transactions, online merchants do not directly receive and store customer information to minimise the responsibility of securing those information as well as the risk of losing customer trust when a security breach occurs. Hence, using a payment gateway is often the solution. In the process of receiving transaction details from merchants and customers as well as forward the transaction results back to the corresponding customers, payments gateways often need to access and store a lot of sensitive information. Typically,

these merchant gateways have dedicated information technology infrastructures to process and store these information securely, alleviating the needs for merchants to perform these tasks. However, using a payment gateway is not an absolutely secure solution to protect customer data, especially when payment gateways can still be targets of malicious activities. There are many known cases of attacks using different exploitative techniques to aim at incompetent online merchants or employees of such payment gateways, making it only partially secure. Finally is the scenario where there is no security at all in an e-commerce transaction. In this case, although customer information sent to merchants is encrypted using Secure Socket Layers, the storage of such information can happen on remote merchant servers or payment gateways without the direct control of customers. Furthermore, data can be stored in unencrypted formats, making them easy targets for malicious attacks. This is quite likely the case, especially when there is an increasing numbers of security breaches in which online merchants or payment gateways have failed to protect the privacy of customers and security of their data.

The above discussion about the different security levels of e-commerce application, i.e. absolute, partial and no privacy, makes it easier to understand the limitations of existing security infrastructure in many stages of e-payment processing. Firstly, to protect the communication between all parties involved in an online transaction, Secure Socket Layer is often used. However, this technique can only protect the information in transit by means of cryptography. At the receiving end, encrypted data still has to be decrypted and processed. It will subsequently be up to the receiving party to decide how to store this information without any effective mechanism for the owner to control such information. This is a limitation because when an owners want to delete their information, for example, they have to do it by sending requests to distant servers outside their controls and trust that those servers will execute their commands unconditionally. Furthermore, the increasing number of data security breaches due to the incompetence of merchants in protecting customer data is an indication that online merchants often retain customer data for various purposes while not providing adequate security protection to such sensitive data. Secondly, many of the existing online payment models are not offering their customers with a truly anonymous shopping experience. There is still a need for customers to buy online items while minimising the amount of information given to online merchants. This will benefit the customers because they can retain as much information as they want while also alleviating the tasks of providing security protection for those information on the merchant side.

6.3 Related Works

The powerful growth of the Internet and e-commerce has created a strong demand for technical solutions to address the security and privacy associated with the use of electronic payment information. One prominent tool that has been proven to be effective is the Secure Socket Layer (SSL) protocol [Wagner et al., 1996], which was proposed in 1994. SSL is designed to have two layers, each of which uses services provided by a lower layer and each layer provide functionalities to higher layers. The SSL handshake protocol is responsible for exchanging cryptographic keys, which include initializing and synchronizing the status of cryptography at two points of a secure communication line. Upon completion of the key exchange protocol, the SSL record layer can be used to send sensitive data. The primary aim of SSL is to provide online merchants and e-commerce organisations with a practical and widely applicable connection oriented mechanism working in the application layer for Internet client and server communication security. It has now become the de facto standard adopted by all parties involved in e-commerce transaction. Its popularity and solid quality has also been in know by many consumers of e-commerce. The confidence of customers will increase significantly if they are ensured that SSL is being used to protect their payment information. However, there are some limitations of SSL, especially when SSL only protects online users against external attacks because the information is only encrypt in transit and need a decryption to process information. Hence, SSL cannot guarantee the customers of an e-commerce website that their online merchants are totally dedicated in keep their sensitive financial data secure in storage.

A very important requirement in any online payment model is how to authenticate customer in different scenarios, especially with and without the presence of a physical card. In the former case, when a customer pays for his goods or services at a store using EFTPOS technology, the authentication service is carried out based on the fact that the customer is possessing a physical card. Additional information such as a signature or pin number of the customer can also be used as solid evidence in authenticating the customer. However, one of the major challenges of online e-commerce is that the payment card is often not present when a financial transaction is about to be made. This is a card-not-present scenario in which the online merchant cannot use the physical card as an proof to authenticate the customer. This case can easily lead to card fraud incidents because it is harder for a merchant to verify the actual card holder before making a decision about the purchase. A common solution would be using other information relating to the card such as card name, address, issuing country, card verification value or even social security number and birthday of the card holder. These details combined together to

prove that the card is authentic and even one of these details is not provided or found out to be incorrect, the merchant has the right to cancel the transaction. However, the limitation of this approach is that customers now have to provide much more information than in the card-present case to purchase online items, making it more likely to compromise the security of their own financial information in cases when security breaches happen at distant merchant servers outside their control.

To address the online privacy risks described above, one research work [Antoniou and Batten, 2011] provides a comprehensive analysis of the concept of trust and personal information distributed in an online transaction from the perspective of a customer. The authors introduce a method to measure trust based on the amount of information given to the parties in the transaction. The paper also provides a theoretical framework to compare online payment processing protocols and some improvements that can be applied to traditional e-commerce models. Another research work [Ruiz-Martínez et al., 2012] focuses on the choices and negotiations required to perform an online purchase, thereby establishing a set of generic components required in creating an online payment processing system. These findings are used as a major tool to create a generic payment protocol for supporting payments with different protocols, including a payment schema and definition of payment extensions to some protocols. The authors also introduce a generic wallet application programming interface to support an electronic wallet for different payment protocols. The proposed model can enhance the trust of customer due to the simplicity obtained from applying these generic components universally in different scenarios to provide a uniform way to make online purchases.

One effective way to authenticate customers when the card is not present is using public key infrastructure. This approach requires a certificate authority trusted by both the owner of the certificate and by the party relying upon the certificate to issue and certify digital certificates. Although this approach can be applied to online merchants, it would be very difficult to authenticate customer using digital certificates because in reality, many online web users do not keep a digital certificate with a pair of asymmetric key. Recent developments of mobile technology have enabled online shopping activities to be carried out on mobile devices [Roy and Venkateswaran, 2014, Chowdhury et al., 2012]. This means that customers will have to keep their private keys on their devices, risking losing the key if their phones were stolen. The limitations of using public key infrastructure make it necessary to consider alternative methods to provide secure transactions. One promising approach is using a password as an additional requirement in the authentication process. It means combining what a customer already possessed, i.e the payment card and all the information on it, with another piece of information

which only the customer knows about, i.e. the password. One of the limitations of this approach is that there is no direct connection between the customer and the card issuing company. Therefore, the merchant or a payment gateway will be responsible for forwarding the customer password to the card issuing company. This is a security risk because a merchant can be dishonest or an eavesdropper can intervene in the intermediate steps to get information about the password. Even if when the password is encrypted by a non-probabilistic cryptographic scheme, the static ciphertext can be used for malicious purposes as it is unchanged throughout many transactions [Butler and Butler, 2015]. However, if an online payment processing scheme is designed so that the password is unique for each transaction, then not only the scheme can minimise the possibilities of replay attacks, but the parties involved in a transaction are also able to determine the expiration time of the transaction.

Another method which is very effective in strengthening the security of online payment processing models is the use of two-factor authentication [Keresman III et al., 2012]. It is a well-known technique using the intersection of two different set of information. These are the sets of information that a user knows about, or something that he possesses, or something that is inseparable from the user. A typical example of two-factor authentication is when a user log in to an online bank account. Not only does the user need to enter a user name and password, but he also needs to wait for the bank to send a security token to his mobile phone and enter that token to the bank webpage to complete the login process. When applied in e-commerce, two-factor authentication technique requires every card holder to carry with them extra hardware devices to complete their online financial transactions [Acharya et al., 2013]. However, it would also mean that customers need to carry many different authentication devices for each of their online merchants. This inconvenience can be reduced if customers choose to register with a third party payment gateway so that they need to use only one authentication device for all their online payments. There are still problems such as when customers are limited to online merchants which affiliate with their selected payment gateway or another limitation that customers still have to carry one authentication device and not complete free to purchase what they want online.

One-time card numbers can also be used to address the security requirements of online payment processing system [Gray et al., 2015, Tan et al., 2014]. A one-time or virtual card number is a randomly generated card number associated with an actual payment card. Different car issuing company allows its customers to set a maximum charge for the virtual card number, which is a very useful feature to protect online transactions. Typically, a customer can set an expiration date to any period of time up to a year from its creation date. Online merchants will

still regard a virtual card number with no difference from a credit card. It means the charges still appear on a regular bill, but the actual card number is inaccessible to the merchants. Another research work [Plateaux et al., 2014] also proposes the use of one-time biometrics for online banking and electronic payment authentication. The authors argue that although user authentication has been enhanced by using two or more factors, there are still plenty of malwares and social engineering attacks making user authentication vulnerable. Their paper proposes the use of biometrics as an additional technique to bolster the security of online banking transactions. Contrary to the standard banking systems, the proposed model requires authentication to be performed by the bank and not only by the user or by the personal device. Their model uses a device called OffPAD to collect biometric data from which one-time passwords will be generated. The authors provide a security analysis to show how the authentication process has been enhanced using their biometric model. Their experiments also show results with high accuracy in terms of false positive. However, their model still requires the use of dedicated devices to generate one-time passwords, which could be an inconvenience when applied to online payment processing system.

6.4 Proposed Model: Secure Payment System With Homomorphic Cryptography

In this section, we will describe the details of our model. This is a new online payment scheme with the ability to preserve the security of data and privacy of customers by a novel application of homomorphic cryptography. Furthermore, not only we want to improve the robustness of e-payment systems, we also want our system to integrate as smoothly as possible to existing online payment architectures. This feature is achieved by the careful analysis of many existing online payment processing workflows when designing our model. Randomisation is a very important part in our model. This functionality is required to help secure the data in transit and storage as well as guard against replay attacks. Hence, we use identification numbers for each invoice and transaction together with timestamps often recorded at the point when a transaction is initiated. Our scheme is designed so that the ciphertexts generated in every transactions are different from each other. Therefore, even if such ciphertexts are recorded in any server of the merchants or any third party involved in the payment model, it would be impossible for a malicious user who has access to those information to retrieve the plaintext or replay the transactions.

Before beginning the description of our model, we make several assumptions regarding the general security infrastructure which exists in any online payment processing system. The first assumption is that the privacy of the communication line is established by the use of Secure Socket Layer (SSL). This is the fundamental security feature of an e-commerce website. SSL is used to secure the details entered by customers when they go to the checkout page and send those details in encrypted format via the public network to either a distant payment gateway or card issuing company for processing. Another assumption is about the existence of trusted parties as a foundation for the basic security features in the first assumption. An example of such trusted third parties is the certificate authority which is responsible for creating and certifying the homomorphic asymmetric key pairs used in our model.

Beginning Transaction. Our online payment model is designed based on existing online payment processing architectures with all the basic components and steps found in many other models, ensuring that our model can be adopted and deployed into existing e-commerce systems. Figure 6.2 shows how our model begins when a customer visits an e-commerce website, adds a few items to the shopping cart and goes to the check out page. At this stage, after checking all the items in the shopping card, the customer click the Check Out button to begin a transaction request. This request will be sent to an e-commerce server to signify the interest of the customer to buy all the items in the shopping cart from that particular merchant. In our model, this request contains a message asking the merchant to send an invoice and the public key of the merchant together with the homomorphic public key of the credit card issuing company.

After receiving this request, the merchant server will begin a process which we call a transaction response. This process includes a reply message sent from the merchant server back to the customer containing an invoice of all items in the shopping cart together with a transaction identifier and two digital certificates from the card issuing company and the merchant. These two certificates must be signed by a trusted certificate authority. The content of these certificates are very similar to an ordinary digital certificate in terms of the number of required fields such as validation time, signature, issuer, etc. In our model, while the merchant public key is an ordinary public key generated by algorithms such as RSA, the key sent by the card issuing company is a homomorphic public key generated by a trusted party using the homomorphic key generation algorithm proposed by The Smart and Vercauteren [Smart and Vercauteren, 2010]. This homomorphic public key not only allows the card data to be encrypted but also enable many types of computing operations such as addition, multiplication and comparison to be executed directly on the resulting ciphertext. A software running on the

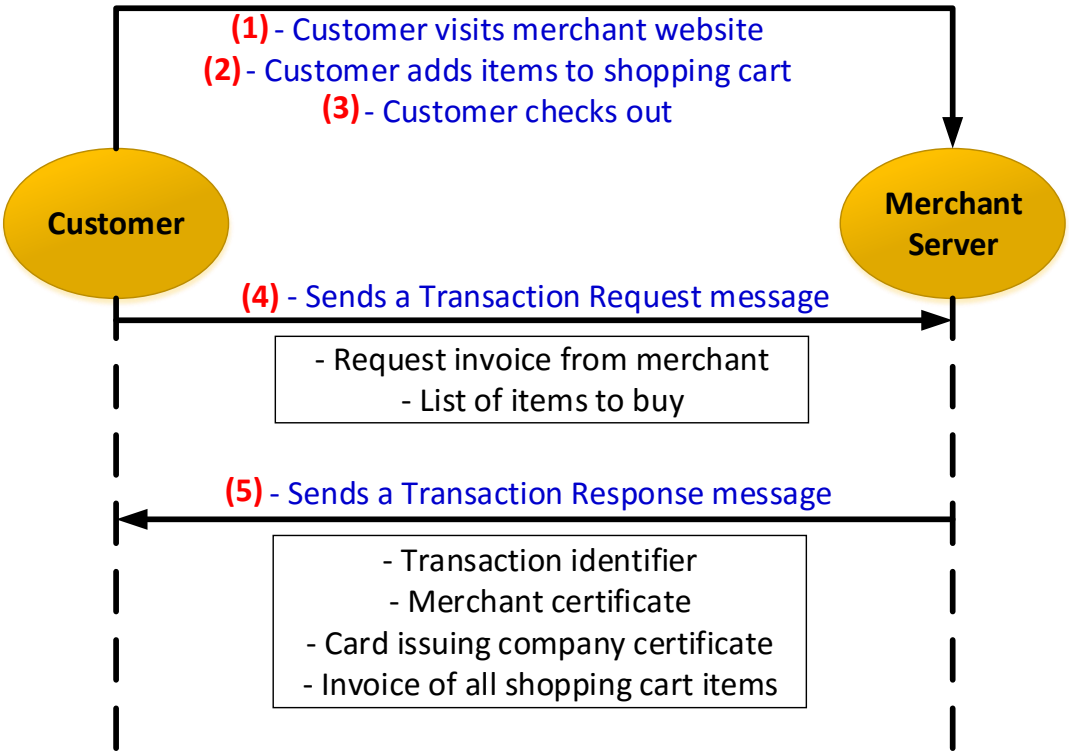


Figure 6.2: The first stage when a transaction begins.

customer computer, i.e. a web browser, will receive the two certificates and validate all the details in these certificates provided by both the merchant and the card issuing company. This is a typical certificate validation process involving all necessary steps such as hash extraction and signature comparison. Finally, if all the details in the certificates are correct, both the customer and merchant in our model will go to the next stage of the online transaction which will be described in the next section.

Payment processing. In the next major step in our online payment processing model, after receiving the public keys of the merchant and card issuing company, the customer will start sending more specific information including details about the items in his shopping cart as well as information about how he is going to pay for the purchase. To protect the privacy of this customer, his information should only be accessible to relevant parties. Any information about the customer that is not necessary for one party should not be accessible to that party at all time. In our model, to finish processing a transaction, the merchant would need to know about details such as the items the customer want to buy, customer name and phone numbers, shipping address and residential address of the customer. The card issuing company needs in-

formation about how the customer would make payment such as credit card details, the name on card, expiration date and the security number on the back of the card. However, the list of items purchased by the customers does not need to be revealed to the card issuing company. Likewise, the merchant does not need to know about the specific payment information such as the card details. Instead, the merchant only needs to know if the payment proposed by the customer has been approved by the card issuing company. In our model, these requirements are addressed thoroughly by the separation of information so that the set of overlapping information received by both the merchant and the card issuing company is minimised.

To achieve our access restriction objectives, we separate information sent by the customer into two groups as shown in Figure 6.3, each information group is stored in a message with minimal overlapping. The first message is called the transaction information which will initially be sent to the merchant, which will forward this transaction information message to the payment gateway and then finally to the card issuing company for processing. Therefore, the content of this message is encrypted using the homomorphic public key of the card issuing company. We define transaction information to be all the necessary details required by the card issuing company to approve a transaction. Hence, this message contains card number, name, expiration date and the card verification value. It also contains a card password known only by the customer, a timestamp and validity period, and finally, an encrypted version of the second message containing the order information. The primary target of the second message is the merchant, hence, it is encrypted using the public key of the merchant. However, a copy of this message with all the fields encrypted by the public key of the merchant is still sent as part of the transaction information message as another method to make this encrypted transaction information message unique. The details included in the order information message are the transaction identification number, the cost of all items to be purchased, the timestamp and validity period. To prevent replay attack, we need a method to specify the expiration of the two messages. That is the reason why the timestamp and validity period fields are included in each of the two messages. They will serve to let the receiver of those messages know the valid time to process the messages. Replay attacks will not be successful after these messages expire because they will be discarded when the time validation checks fail.

To protect our online payment processing scheme against replay attacks, we need to design our scheme so that after encryption, the ciphertexts of both the transaction and order information messages are unique. However, we do not want to rely only on the probabilistic feature of the homomorphic cryptography scheme we are using. Therefore, we add additional information to create uniqueness. In addition to the fields required for financial processing, the transaction

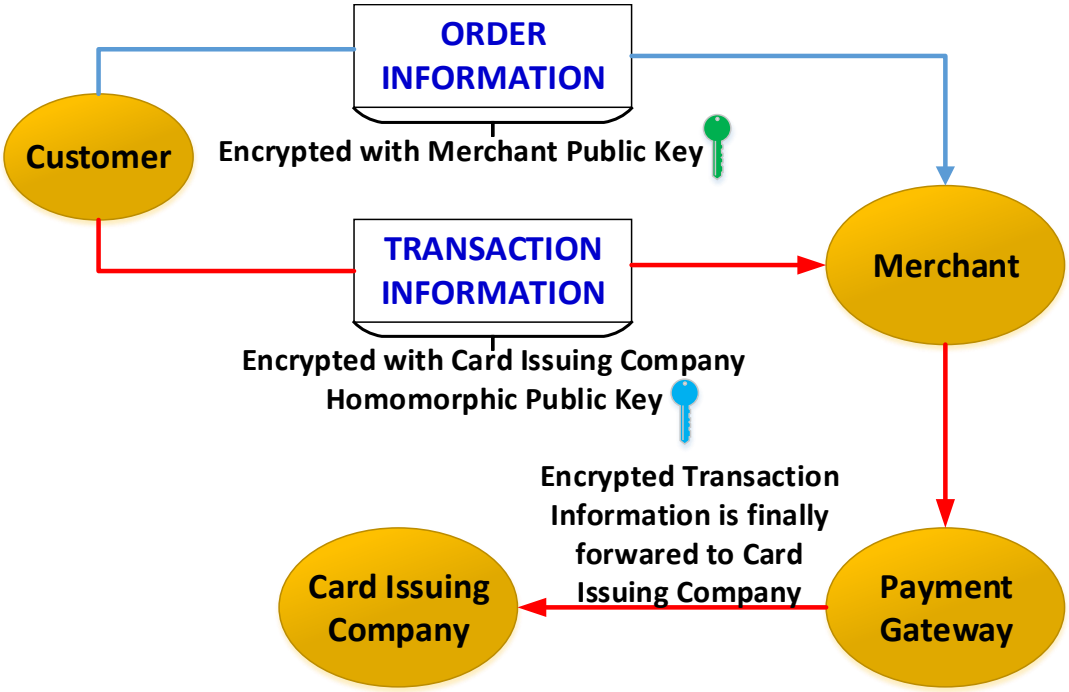


Figure 6.3: The order information message is encrypted with the merchant public key and is sent to the merchant. The transaction information is encrypted with the homomorphic public key of the card issuing company which will finally process the data directly on encrypted domain.

information message also contains the card password, a timestamp and the encryption of the whole order information message. We also include the homomorphic encryption of the timestamp value because it is very effective in ensuring the one-time usage of the two encrypted messages even in cases when a malicious users can obtain the timestamp of the transaction. One problem we encounter with this design decision is how to compare the ciphertext of the order information message with the card details stored in the servers of the card issuing company. They are both in encrypted form and the comparison must be performed directly on the ciphertext to maximise the security of user data. Finally, we decide not to incorporate the timestamp field in the encryption of the card number and use the encrypted card number as an identification mechanism to map the encrypted details provided by a user to the encrypted profile of this user already stored on servers. We include the timestamp in other fields, except

the card number field, as shown below:

$$\begin{aligned}
\mathrm{c}_1 &= \mathrm{E}_{\mathrm{public_key}}(\mathrm{E}_{\mathrm{public_key}}(\mathrm{cardHolderName}) + \mathrm{timestamp}) \\
\mathrm{c}_2 &= \mathrm{E}_{\mathrm{public_key}}(\mathrm{cardNumber}) \\
\mathrm{c}_3 &= \mathrm{E}_{\mathrm{public_key}}(\mathrm{E}_{\mathrm{public_key}}(\mathrm{expiryDate}) + \mathrm{timestamp}) \\
\mathrm{c}_4 &= \mathrm{E}_{\mathrm{public_key}}(\mathrm{E}_{\mathrm{public_key}}(\mathrm{orderInfoMsg}) + \mathrm{timestamp})
\end{aligned}$$

From the encryption examples above, only the card number is encrypted directly using the homomorphic public key without the timestamp, so that when its corresponding ciphertext, i.e. c_2, is finally sent to the card issuing company, a homomorphic comparison operation will be performed directly on encrypted domain to determine if c_2 is identical to the stored ciphertext of the card number. For the other fields, the homomorphic public key of the card issuing company will be used to encrypt that field first, then the resulting ciphertext will be encrypted again together with the timestamp to add an extra layer of randomization and security. The validation of customer information in the card issuing company will be performed directly on encrypted domain and can be summarized as followed:

1. When issuing a card to a customer, the card issuing company also stores information about that card on its servers. Specifically, a field F, which can contain values such as name of the card holder, the card expiry date, the card verification value, etc, is encrypted using the homomorphic public key of the card issuing company and stored in the following format:

$$\mathrm{E}_{\mathrm{public\_key}}(F)$$

2. When the card issuing company receives a new transaction request, this request will be followed by the encrypted transaction information message containing all the encrypted fields sent from customer and also the time stamp. We call c_{received} the ciphertext that the card issuing company receives. This ciphertext was created on the client side using a piece of card information D provided by the customer. The card issuing company will check if D is identical to F on encrypted domain. To do so, the card issuing company will encrypt the time stamp with the encrypted version of F that it stores to obtain a new ciphertext called c_{computed}. Both c_{received} and c_{computed} are calculated using the following equations:

$$\begin{aligned}
\mathrm{c}_{\mathrm{received}} &= \mathrm{E}_{\mathrm{public_key}}(\mathrm{E}_{\mathrm{public_key}}(D) + \mathrm{timestamp}) \\
\mathrm{c}_{\mathrm{computed}} &= \mathrm{E}_{\mathrm{public_key}}(\mathrm{E}_{\mathrm{public_key}}(F) + \mathrm{timestamp})
\end{aligned}$$

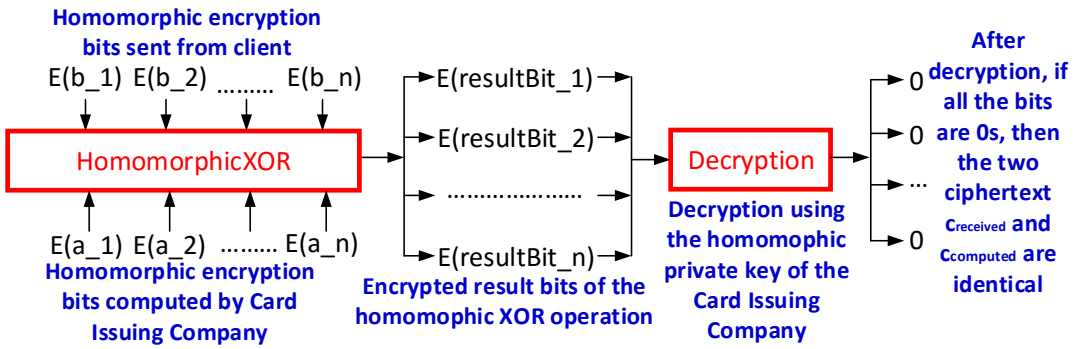


Figure 6.4: The homomorphic comparison process is performed over each encrypted bit and output encrypted results which will be decrypted by using the homomorphic private key .

3. The homomorphic cryptography algorithm that we use will compute and represent c_{received} and c_{computed} as two arrays of encrypted bits. This homomorphic cryptography algorithm also allows a homomorphic XOR operation to be performed on two homomorphically encrypted bits. Therefore we use this operation as a basis to check the credit card information directly on encrypted domain. Each encrypted bit of the c_{received} ciphertext and the corresponding encrypted bit of the c_{computed} ciphertext will be passed to the homoXOR function as in the following equations:

$$E(\text{resBit}_1) = \text{homoXOR}(b_{1c_{\text{computed}}}, b_{1c_{\text{received}}})$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$
$$E(\text{resBit}_n) = \text{homoXOR}(b_{nc_{\text{computed}}}, b_{nc_{\text{received}}})$$

   Each encrypted resulting bit will then be decrypted using the homomorphic private key of the card issuing company. If all zeros are obtained, then D and F are identical. However, if only one bit is 1, then D and F are not the same and the card issuing company stop the homomorphic check and return a rejection message to the payment gateway which in turn will notify the merchant to cancel the transaction. After the card issuing company check that all the encrypted values provided by the customer are authentic, then it will notify the bank and the payment gateway, which in turn will notify the merchant. Figure 6.4 illustrates this homomorphic checking process.

The next design decision we need to make is how to provide consistency in the use of timestamp between all the parties involved in one transaction. The timestamp is created when the transaction is started and used by the customer to encrypt all the fields in the two messages. If the time is synchronised correctly, than all parties participating in the transaction will use the

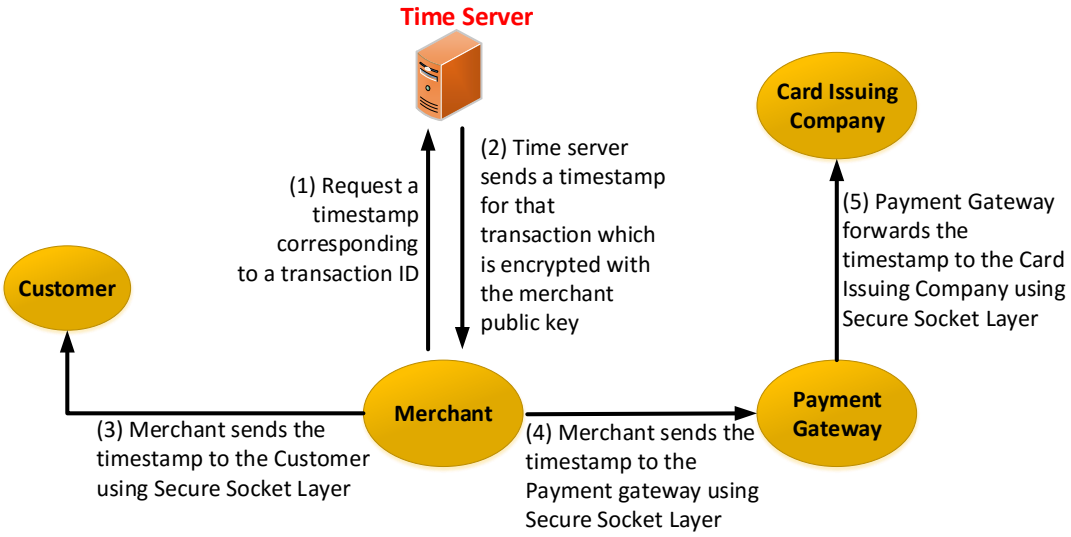


Figure 6.5: A trusted time server will generate a timestamp for each transaction and send to the merchant in encrypted form using the public key of the merchant. The merchant server will distribute the timestamp to other parties using Sercure Socket Layer.

same timestamp, producing accurate result when performing homomorphic comparison directly on ciphertexts. However, if the timestamp is not kept in sync by one party, for example, the customer or the card issuing company or the merchant, then the homomorphic comparison result will be totally different, causing the transaction to be cancelled by the merchant or the card issuing company. Therefore, in our model, the timestamp creation and allocation can be managed by a trusted generic time server which is known to both the merchant, the card issuing company and the payment gateway, as shown in Figure 6.5. When a transaction is initialized, the merchant will request the generic time server to issue a timestamp corresponding to that particular transaction. To do so, the merchant will also include its public key, its identification numbers and the transaction identification number in each timestamp request sent to the trusted generic time server. The trusted time server will generate a timestamp and stored that timestamp in its database with the identification numbers of the merchant and that transaction as the key set of that particular record. The generic time server will encrypt that timestamp with the public key of the merchant before sending the encrypted timestamp to the merchant which will decrypt using its private key. That timestamp will be sent to the other parties such as the client, the payment gateway and card issuing company via encrypted Secure Socket Layer connection established when the merchant sends other data to the other parties. We designate the merchant as the only party responsible for sending the timestamp when the

customer sends a request for purchase of all the items in the shopping cart. As an alternative, the certificate authority can also provide a universal timestamp for all the parties, especially when the client web browsers verify the authenticity of all public keys.

After receiving the transaction information and order information messages sent from the client, the merchant will process the order information message. The first step carried out by the merchant is decrypting all the fields in the message separately using the private key of the merchant. All information relating to the purchase will be checked by the merchant against the invoice sent earlier. If the merchant finds any inconsistency in the invoice and the item costs sent from customer in the order information message, the merchant will stop processing the transaction as well as stop forwarding the payment information message to the payment gateway or the card issuing company. If all the integrity and expiration checks are passed, the merchant will generate a unique payment identification number for this transaction and forward the payment information message to the payment gateway for the next payment processing stage. The payment gateway will verify and check the integrity of the transaction information message before sending it to the card issuing company for payment processing. The merchant will also send some information regarding its identity such as account details and merchant certificates, etc, to the payment gateway which will later forward these details to the card issuing company for identification and authorisation purposes. One of our features to increase the flexibility of the entire payment system architecture is that the merchant can choose to communicate with the payment gateway directly or it can also choose not to use the service of the payment gateway to communicate with the card issuing company directly.

When the card issuing company receives the transaction information message sent from the payment gateway or directly from the merchant, it will begin the verification and processing step. Instead of decrypting the message to get the payment information, the card issuing company will perform a homomorphic comparison directly on encrypted data to check if the encrypted card information is identical to the encrypted information stored on its server. These homomorphic operations will involve comparison of the encrypted card number, name on card, expiration date, card verification value and the encrypted card password known only by the holder of the card and its copy stored in encrypted format on the servers of the card issuing company. The results of these comparison operations are also encrypted and can only be decrypted by using the homomorphic private key of the card issuing company. If one of the comparison operation fails, it means that at least one payment details provided by the customer is not authentic and the card issuing company has the the right to disapprove the transaction. However, if all the comparison operations prove that the customer has sent valid payment

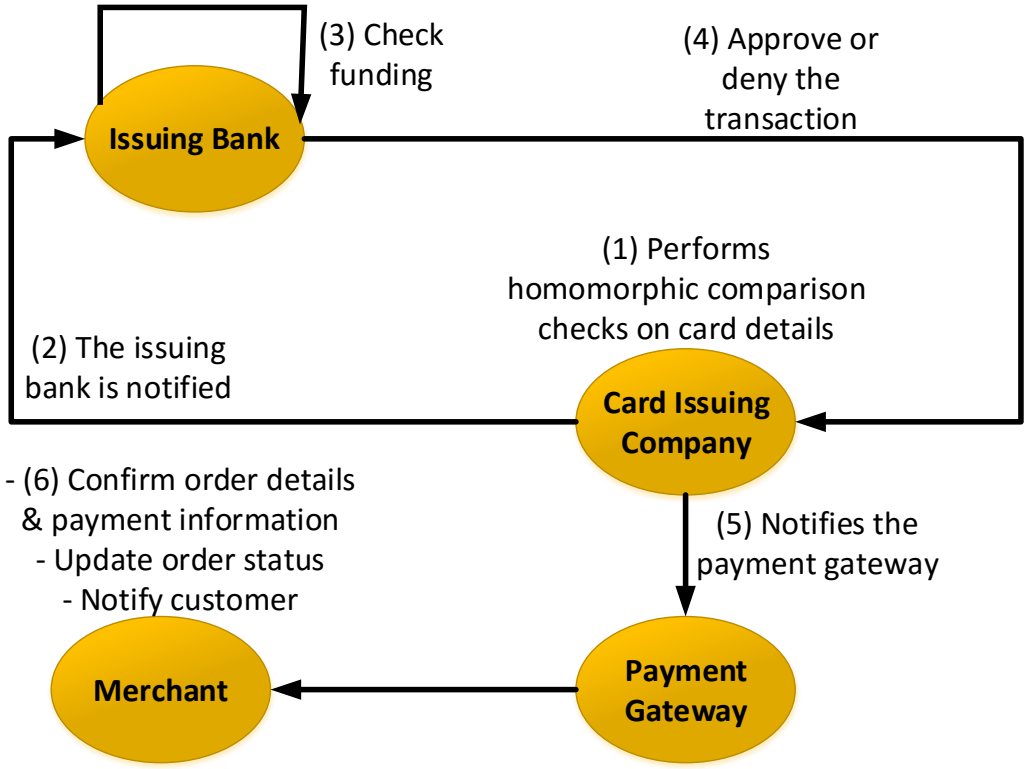


Figure 6.6: The finishing stage of a transaction.

information, then the card issuing company will notify the issuing bank, which in turn will check if there is enough fund to pay for the transaction and send a message to the card issuing company to approve or deny the transaction.

After receiving a message from the issuing bank regarding the availability of funds for that transaction, the card issuing company will notify the payment gateway, which will process the payment. The online merchant will finally receive a notification from the card issuing company. The merchant knows what order to process because the encrypted order detail is also sent from the card issuing company as a confirmation. After confirming the order detail and payment information, the merchant will update the order status and notify the customer, usually by sending emails, to let he know about the success or failure of the request for purchase. The whole process is illustrated in Figure 6.6.

6.5 Experiments and the Security of Our Model in Various Scenarios

To measure the performance of our model, we performed experiments with different number of homomorphic encrypted bits to measure the computation time required for the homomorphic comparison operations on encrypted domain. The user of our application can choose a variable number of encrypted bits for each encrypted field in the transaction information package. Alternatively, users can also use fixed-length encrypted field by first passing all plaintext values to a hash function before applying homomorphic encryption. In our experiments, we use a variable number of encrypted bits, i.e. 8 bits, 16 bits, 32 bits and 64 bits to represent each encrypted field in the transaction information package processed by the card issuing company. These variable number of bits used in the encrypted fields will give us an accurate way to measure the computation time required to perform homomorphic comparison on encrypted domain.

Our experiments were conducted on a machine having 4 Gigabytes of memory and 2.4 Gigahertz quad core processor with Ubuntu Linux as the operating system. The homomorphic cryptography algorithm we use will encrypt a bit into a large integer. Therefore, each encrypted field is represented as a vector of large integers with each of the elements of the vector corresponding to each encrypted bit. Table 6.1 shows the results of our experiments. As the number of bits used in each encrypted field increases, the homomorphic encryption time, homomorphic XOR and decryption time also increase.

Next, we measure the performance time of our system when there are large numbers of comparison operations going on, especially when the card issuing company has to process a large number of transactions at a time. In each experiment, we chose a large number of comparison operations which are executed when different numbers of bits are used to store the encrypted fields. Table 6.2 shows the results of our experiments. Our experiments show that the larger the number of comparison operations, the longer it takes to perform the computation operations on encrypted domain. However, in reality, these computing operations are not performed sequentially and the computation time can be reduced by distributing the computing workload to different servers running in parallel.

The objectives of our secure online payment processing model is to enhance the security of e-commerce businesses as well as to protect the benefits and privacy of both buyers and sellers across the Internet. However, there are also many adversary scenarios in which malicious users are discovering new methods to obtain illegitimate advantages at the expense of other

Table 6.1: Measuring homomorphic comparison time when variable numbers of encrypted bits are used for each encrypted field in the transaction information package processed by the card issuing company

Number of Encrypted Bits	**Homomorphic Encryption Time (ms)**	**Homomorphic XOR Time (ms)**	**Homormophic Decryption Time (ms)**	**Estimated Computation Time (ms)**
8	97.92	26.24	125.76	249.92
16	195.84	52.48	251.52	586.73
32	391.68	104.96	503.04	1078.66
64	783.36	209.92	1006.08	2956.85

Table 6.2: Measuring homomorphic comparison time when there are a large number of comparison operations applied to a variable number of bits used to represent the encrypted fields in the transaction information packaged processed by the card issuing company

	Total Computation Time Corresponding To a Number of Bits (unit: seconds)			
Number of Comparison Operations	**8 bits**	**16 bits**	**32 bits**	**64 bits**
500	124.96	249.92	579.84	1072.98
1000	249.92	576.84	1836.68	2495.59
5000	1249.6	2499.2	4998.4	11035.58
10000	2576.84	5270.16	13769.58	23515.67

ordinary online users. This is due to the fact that in online e-commerce environment, the design of many payment processing models can have loopholes which can give both customers and merchants the opportunities to behave maliciously, for example by simply not following a protocol, or withholding or changing some information in intermediate steps. Therefore, although a proposed model has defined clearly its transaction protocol together with all the steps that participants need to follow, designers of such protocol need to think outside the box to account for all the situations that might possibly go wrong due to the deliberate or unintentional misbehaviours of participants of such protocols. To ensure that e-commerce will continue to be used by an ever increasing number of users, it is essential to carry out such analysis. This process will also help to make it clear about the participation of merchants in protecting the security of their online model, especially when an analysis can help all the parties to see the causes of a security breach. In this research work, we experiment with our online payment processing model and show how its security can be tested under many adversarial

situations described below:

- Scenario 1: Information of a customer is replayed by a malicious party after a transaction has finished.

  In any online payment processing model, a significant amount of information is sent from a customer to another party, which is most often to the merchant, to initialise and authorise the transaction. Therefore, it is the responsibility of the merchant to keep this information safe from the harmful actions of malicious parties. On the other hand, there are additional risks from the merchant itself. For example, when the online business organisation is not a trusted entity, or employees working inside can get unauthorised access to customer data. These factors make it possible for fraudulent transactions to be carried out in which financial information may be replayed after the issuing bank of the card holder has approved the original transaction. However, in our model, all the packages such as the order information and transaction information are encrypted by the homomorphic public keys of the merchant and the card issuing company respectively. Furthermore, in the encryption process, random information such as the security token and timestamp are also included in the content of such messages. Therefore, if a replay attack was to happen, the card issuing company could verify the timestamp and the security token of the message and choose not to authorise if such random information was found to be expired.

- Scenario 2: Dealing with a middle attacker spying on an approved payment status issued by a payment gateway.

  When the online merchant receives an approved payment status from the payment gateway, it will proceed with the transaction and start delivering the goods to customer. However, if there is a middle man who can intercept the communication line and get the content of the approved payment status, he can then replay the message at a later time, making the merchant to deliver again the goods that it has already delivered. In our model, this scenario is addressed effectively through several technical solutions. First, each transaction is assigned a unique ID by the merchant, which will also update it database immediately after receiving an approval or denial result for the payment authorization status from the payment gateway. Furthermore, there is an encrypted timestamp field included in the order information that can only be decrypted by the merchant to

verify the expiration time of the message. These details combined together will make it very difficult, if not impossible for an attacker to replay the approved payment status at a later time because the merchant will refer to the database or check the timestamp and validity period of the message to know instantly whether it has already received the approval or denial status of the transaction from the payment gateway.

- Scenario 3: A dishonest customer tries to send several refund requests for one returned order.

  In a typical transaction, a customer will receive all items in the order from the merchant after his payment is approved by the payment gateway. However, if the customer does not satisfy with the goods he has purchased, he is able to send them back together with a refund request. Our model is required to protect merchants in case a dishonest customer tries to send several refund requests for one returned order. We need to have some data for all parties involved in an order kept and to refer to in case requests for refund is sent from customers. Therefore, when a merchant receives a refund request, the authorization status sent from the payment gateway can be used to determine the actual amount debited previously. After calculating the exact amount to be returned to the customer, the merchant will send that amount together with the authorization status to the payment gateway. The gateway is responsible for handling the details of the refunding task, including debiting the fund from the merchant's account and deposit it into the account provided by the customer. At this stage, the authorisation status is the one piece of information that is visible and can be referred to by all parties involved in the refund request such as the card-holder bank, the payment gateway, the merchant and the customer. Therefore, the possibility that a malicious party claiming more money than the actual amounts paid for the product is reduced significantly, if not eliminated completely because no party is able to generate a relevant authorization status for the other parties. The authorization status also prevents dishonest merchants to deny a product purchased by customer because the customer can prove that to an authority and compare his authorization status with those from other parties to determine if a merchant has denied an order.

In addition to addressing various use case scenarios mentioned above, our online payment processing model also has many outstanding characteristics that worth analysing in detail. Most importantly, the homomorphic cryptography scheme used in our model helps the mer-

chant, card issuing company and payment gateway to check customer details directly on the ciphertexts without compromising the security of the encrypted sensitive customer data, protecting confidentiality and enhancing efficiency. We have separated the transaction data into two packets, transaction and order information which are encrypted separately by the merchant and the card issuing company. No third party in the communication line can decrypt these packets without a valid decryption key. Furthermore, comparison results are performed directly on ciphertexts, reducing the computation works to decrypt the messages to help our model more scalable. There are also non-reusable factors applied in our models, which are a timestamp and a random security token when encrypting the payment information. Therefore, the information provided by the customer to the merchant will become obsolete after a transaction has ended, making it very difficult for a man-in-the-middle or replay attack as long as all the private homomorphic keys are protected by the merchant and the card issuing company. It would also mean that the customer is having full control of their information because their encrypted sensitive information will expire after it has been used for the first time, making it impossible to reuse this encrypted information even when a malicious party is recording the data along the communication line.

In any payment processing model, it is very important to minimise the information given by the customer to the merchant or a payment gateway so that the anonymity of customer is protected while still allow the merchant to verify authentic payment information. In our model, this requirement is achieved by combining two pieces of security information that the customer controls, i.e. what the customer knows, which is the card password and what the customer possesses, which is the card details and security code. The card issuing company is the main agent responsible for authorizing the customer details forwarded from the merchant by using homomorphic cryptography. It will use homomorphic comparison to compared directly the stored ciphertexts of the card number and card password with the given ciphertexts. Finally the merchant will be notified with an absolute degree of certainty if the customer is actually the owner of that card. All of this process is performed while the payment details are kept secret from the merchant because of the separation of information and the use of homomorphic cryptography. Our model is also efficient when carrying authentication of a customer because we do not require customer to log in to a web page or a payment portal in order to verify themselves. Instead, a customer can use a password associated with a card when providing payment details. Finally, our protocol is built upon existing e-commerce payment models, making it easier to implement our model without significant changes to the current payment system of an online merchant. Customers can begin to use the model immediately after receiving their

card because there is no requirement to perform any further registration with a third party certificate authority.

6.6 Conclusion

In this research work, we have described in detail the design and components of a secure online payment processing model with the ability to protect sensitive customer financial information. A prominent feature of our model is the separation of transaction information into encrypted packages only accessible to relevant parties involved in a transaction. In our model, homomorphic cryptography is used not only to enhance the security of encrypted packages, but also make it possible to process payment information directly in encrypted format. The non-reusability of encrypted data in our model is also guaranteed by the use of random factors such as unique time-stamps and random session keys, making sensitive information sent out by customer obsolete after the first use. This is an important mechanism to guard against many security risks such as a replay attack or an identity theft. We have also discuss how our model can still perform robustly in many different adversarial scenarios. In the future, we plan to expand and add more security features to our model so that it can be applied in larger e-commerce applications with higher security and efficiency requirements.

Chapter 7

Conclusion

Homomorphic cryptography is an effective tool for data owners to control the security of their cloud-based data. The reason is because when individuals or business organisations start using cloud services, they often do not have full control of their data storage and computation because most of the physical data and computing infrastructures are managed by cloud service providers. Without the use of cryptography, security breaches are likely to happen because data owners do not have other mechanisms to control the security of their data. Furthermore, even with the use of strong cryptographic schemes, the cloud will become a place to store encrypted data only, while many of the data processing tasks must still be performed on the client side after the encrypted data from the cloud was downloaded and decrypted. This limitation can affect performance, especially when the client side has limited computing resource such as when a mobile phone is used or when there is a large amount of data to be processed.

To create a more powerful and flexible cloud-based data storage models, not only data must be stored in encrypted form, but such models should also provide the ability to compute on encrypted data. Specifically, authorised parties in those models can request the cloud to perform many computing operations securely on encrypted data. At any stage in the cloud-based computation process, the confidentiality and privacy of the data are always protected because no decryption is required. The output of such computing operations will also be encrypted and can only be decrypted by authorised parties.

In this thesis, we propose more effective and efficient cloud-based data storage and computation models with the application of homomorphic cryptography. Our models provide authorised parties with the capability to request the cloud to perform many computing operations securely on encrypted data. Unique features of the homomorphic cryptography scheme

applied in our models allow the privacy and confidentiality of the encrypted data to be protected because decryption will not be required during the computation process and the outputs of such operations are also in encrypted forms to be decrypted only by authorised parties.

7.1 Concluding Remarks and Discussion

One important characteristic to determine the success of a cloud computing model is how it can provide a strong security to both data stored on the cloud and to all computing operations performed over the encrypted data. User privacy should also be protected at the lowest levels during the execution stage of a protocol. In our research works described in previous section, we provide answers to important questions about the design and implementation of cloud-based computing models which can be used for storage and computation of encrypted data while being able to protect data security and preserving the privacy of the users. We always think about the applicability and practicality of our proposed models, hence, our research works presented in the four core chapters are not only about how homomorphic cryptography is used, but they are also about the efficiency and effectiveness of the new cryptographic paradigm in such real life scenarios. Some of the main findings of each core chapter are discussed in detail below.

7.1.1 Design and Implementation of a Secure Cloud-based Billing Model for Smart Meters as an Internet of Things Using Homomorphic Cryptography

In this research work, we have identified and proposed a solution for several security and privacy issues relating to the data exchange, storage and processing on a smart grid. Such a grid often has various types of energy monitor devices connected directly to smart meters via wireless links to help consumers to view their usage history, the amount of electricity they are using, and the current tariff. However, there are many outstanding security and privacy issues, especially when smart meters are connected to public networks. There is a strong possibility that various usage patterns of users can be tracked using load monitoring techniques to process smart meter data travelled back and forth without using any form of encryption. Another issue is that complex computing tasks over energy usage data can not be performed directly on smart meters because they are constrained devices with limited hardware capability. With the support of the cloud, complex computing tasks such as billings and analysing smart meter

data can be performed quickly and efficiently as long as the user privacy and security of their data are guaranteed. Therefore, we conducted this research work to enhance the smart grid with all desired features offered by cloud computing while maintaining the confidentiality of smart meter data and the reliability of the grid's information infrastructure. Our contributions can be summarised as follows:

- Our model stores and processes smart meter data directly on the cloud while protecting user privacy and data confidentiality using asymmetric homomorphic cryptography. The decryption of user data and results of the cloud-based computation can only be decrypted by parties possessing the decryption keys. We designed our model so that household owners can decrypt their smart meter readings when they want information related to their total data usage and other statistics such as daily or hourly consumption. The grid operator can also access the fine-grained meter readings to monitor the performance of the grid. Data aggregation results are accessible to the energy retailers because they only need to know the total usage in a month or a quarter.

- The application of homomorphic cryptography in our model enables the cloud to compute the aggregated energy consumption of a customer in a given period of time. This task is accomplished because the cloud can perform a series of homomorphic addition operations over an arbitrary number of encrypted fine-grained readings sent from smart meters. Data confidentiality is maintained throughout this process because the cloud does not need to perform any decryption when aggregating encrypted data.

- Our model also provides retailers with the option to offer consumers a range of flexible pricing policies based on the time-varying costs of electricity procurement at the wholesale level. Retailers can request the aggregated energy consumption of their customers during different time periods. After receiving such requests, the grid operator will perform most of the data aggregation tasks directly on the cloud before decrypting the results and sending them back to retailers.

- To improve the performance of our model, we propose a parallel algorithm to aggregate encrypted smart meter readings. This improved model makes used of the processing capability of multi-core processors in cloud servers so that the computation time can be reduced significantly compared to using our sequential homomorphic addition algorithm. To evaluate the enhanced performance of our parallel algorithm, we used Amdahl's law and a metric called speedup. We show how many times faster our parallel algorithm can

actually run on some common processors with different number of cores and threads. We also demonstrate by many examples how an increase of speedup is determined by the size of the homomorphic expression as well as the number of threads used for the parallel computation process.

7.1.2 Secure Pricing Comparison Model Using Homomorphic Cryptography

In this chapter, we describe our novel application of homomorphic cryptography in designing and implementing an approach to securely compute, compare and check the integrity of results in a common e-commerce price checking model. The application scenario begins when a customer visits an e-commerce website and wants to buy an item. Normal procedure requires the client to add his selected item to the shopping basket, then check out and provide his card numbers as well as other personal details such as phone numbers and address. However, a limitation of the traditional approach is that clients have no control over their personal details once those information is stored and processed in distant servers. Many e-commerce websites also want to ensure that a client has sufficient fund before approving the purchase. Therefore, we propose a new secure approach to e-commerce which allows the client balance to be compared with the actual item cost while protecting the confidentiality and privacy of the data owner. The chapter has the following contributions:

- We applied homomorphic cryptography to securely compare integers in encrypted domain using the ciphertexts of their corresponding binary digits while still ensuring client privacy because most of the computing operations are performed on encrypted data without decryption of intermediate steps.
- Our model also reduces the use of a trusted third party by proposing a new mechanism based on homomorphic cryptography. Our mechanism enables clients and serve to validate their transaction outcomes reliably and securely.
- With the user of small ciphertext and key size while performing homomorphic computing operations directly on encrypted binary numbers, our implementation has achieved the desired efficiency and flexibility. We also design our scheme so that an arbitrary number of parties can join the homomorphic cloud-based computation.

7.1.3 Cloud-based Homomorphic Secret Sharing and Access Control

This chapter describes our solution to address the security and access control requirements in large business organisation where the access to highly sensitive data might be controlled by a group of users rather than a single user. We apply homomorphic cryptography to propose a model that allows members of the group controlling access to jointly authorise a set of access rights to encrypted documents stored on the cloud. We address one of the limitations of Shamir's secret sharing scheme which requires a trusted party to collect and combine shares of the encryption key before performing computation works to recover the original key. Our model allows the key sharing model to be executed on the cloud in which the main computing agents are semi-trusted cloud servers located outside the business organisation. Our new homomorphic secure secret sharing scheme can be summarised in the following contributions:

- We propose a new homomorphic key distribution model which allows business organisation to outsource data storage and computing tasks to the cloud in the most secure way possible while ensuring user privacy and data confidentiality.
- Another important feature of our model is how a group of users can use our homomorphic secret sharing mechanism to create a shared secret on the cloud which executes most of the computing task to create, distribute and recreate the shared secret. Each member of the group is able to keep his or her share of the secret private while still being able to combine these shares on the cloud.
- To demonstrate the applicability of our homomorphic key distribution and secret sharing model, we implement a new access control paradigm so that a group of users can control access to encrypted documents stored on the cloud. Each group member can authorise access by submitting their encrypted shares of the secret to the cloud without compromising the security of all secret shares possessed by members of the group controlling access as well as any encrypted documents stored on the cloud.

7.1.4 Secure one-time e-commerce transactions using homomorphic cryptography

In this chapter, we propose a solution to overcome the limitations of the payment gateway in online e-commerce systems. Security issues often encountered by a payment gateway are insider attack, specifically, how to trust their employees in protecting customer information of many

online businesses using their services. There is an increasing number of major security breaches involving third party payment service providers, compromising the data security of customers participating in e-commerce transactions. We address the demands of online merchants to have more secure data storage and transaction models which give customers direct control to protect the security, integrity and privacy of their data from all other parties participating in online transactions. Therefore we propose a novel online payment system using homomorphic cryptography to allow goods and services to be purchased by any customers of online merchants without actually revealing their sensitive and personal details. Our model has an effective feature to enhance customer privacy by making payment details sent to online merchants to expire after the first use. This is very useful, because when an online merchant or a payment service provider lost control of their customer data, no one else will be able to reuse those information for illegal purposes. Our research work can be summarised in the following points:

- Our payment system uses homomorphic cryptography to protect data confidentiality and the privacy of customers participating in online transactions. These features are achieved because our model requires only the minimal set of personal information from customers to provide to merchant servers. Data is processed securely on encrypted domain so that replay attacks by a middle party attempting to reuse such data will not be successful. Customers will get the full benefits from our security features while merchants are supported by our model so that their tasks of protecting sensitive customer data will not incur any additional computational cost.

- We propose a mechanism to compartmentalize sensitive data into different encrypted packets which are only accessible to relevant parties in a transaction. This is the key feature to maintain data confidentiality due to the use of homomorphic cryptography. Customers using our model can keep the items that they are buying secret from irrelevant parties such as the card issuing company or the payment gateway while the merchants also cannot access payment information of customers that they are not supposed to see. No dishonest or malicious users can intercept the communication line and decrypt the encrypted packets at any given time as long as the private keys of the merchants and the card issuing companies are adequately protected.

- Another important feature that we have achieved in our model is the non-reusability of encrypted sensitive customer data. We use a unique time-stamp and random session key when encrypting and executing important customer data. In every request for payments

made by customers, the ciphertexts of the payment details will be unique, making it impossible to use such ciphertexts more than once. This important feature helps our model to avoid many security risks such as a replay attack or an identity theft.

7.2 Future Work

This thesis has focused on new cloud-based applications so that data can be outsourced and processed directly on the cloud while maintaining the data confidentiality and user privacy throughout the processing stages. The models and applications that we have proposed so far make use of homomorphic cryptography as the key technology to overcome the limitations of existing research works by processing encrypted data directly on the cloud. The readers of this thesis have seen how homomorphic cryptography has been very useful in the design and implementation of cloud-based data storage and processing model which can preserve user privacy and the confidentiality of smart meter data on a smart grid. This new cryptographic paradigm has also been applied to create a privacy-preserving price checking model allowing an e-commerce server to check if a client has enough money to buy one of its products without knowing the actual balance of the potential client. We also show in this thesis how homomorphic cryptography can be combined with secret sharing to create a useful application enabling a group of users to control access to encrypted documents stored on cloud servers. And finally, the potential of this cryptographic technology is proven again in the design of a secure online payment processing model with the ability to protect sensitive customer financial information. While these tasks have been successfully completed with high performance achieved in the proposed models, there exists some other possibilities that haven't been the focus within the scope of this research. However, addressing these issues could considerably add value and improve on some of the limitations in the developed models, opening up pathways for future research based on the context of this thesis. These can be discussed as follows.

- Although homomorphic cryptography has proven to be the indispensable technology in many secure applications proposed in this thesis and from the research community, the performance of such cryptographic schemes still need to be improved to make it more scalable. This is a crucial requirement for cloud-based applications because of the large number of users as well as a range of processing tasks that can be performed on cloud. Therefore, we will work on more efficient methods to allow the cloud to further reduce the

homomorphic computation time as well as more efficient and scalable cloud computing services on encrypted data.

- After a long time doing extensive research with homomorphic cryptography, we also realise the potential of using homomorphic cryptography in so many other areas such as data mining and analytics. Because the amount of data being collected by many governments and business organisations are growing exponentially, the demands to extract meaningful information from such huge data repositories are also increasing. However, these data mining tasks can only be possible when an application model can ensure user privacy and confidentiality of such data. We know that the most effective way to meet such demands is using cryptography, especially homomorphic cryptography because of its ability to allow processing tasks to be executed directly on encrypted domain. Therefore, in the future, we also want to we will use this approach to build secure data mining applications such as secure k-mean clustering schemes that require no decryption of intermediate homomorphic computation results. We believe this is the next major direction of this interesting cryptographic paradigm in an application context where cloud and mobile computing are the dominant technologies.

- In the future, the important tasks that we want to achieve is improving the existing functionalities and adding more features to our homomorphic-based processing applications running on cloud. For our secure smart meter billing model, we want to add more billing options to support different types of discounts and pricing policies as well as optimize the homomorphic data aggregation algorithm to reduce processing time. For the secure e-commerce price checking model, we will improve our comparison checking algorithm so that the number of check bits can be reduced or increase the accuracy of identifying a failed check. Our secure homomorphic cloud-based secret sharing scheme can also be improved with arbitrary number of members in the group controlling access to encrypted documents. And finally, with our secure one-time e-commerce transaction model, we plan to expand and add more security features to our model so that it can be applied in larger e-commerce applications with higher security and efficiency requirements.

Bibliography

S. Acharya, A. Polawar, and P. Pawar. Two factor authentication using smartphone generated one time password. *IOSR Journal of Computer Engineering (IOSR-JCE)*, 11(2):85–90, 2013. Cited on page 124.

G. Acs and C. Castelluccia. I have a dream!(differentially private smart metering). In *Information Hiding*, pages 118–132. Springer, 2011. Cited on pages 40 and 50.

N. Aes. Advanced encryption standard. *Federal Information Processing Standard, FIPS-197*, 12, 2001. Cited on pages 65 and 86.

M. Akbar and D. Z. A. Khan. Modified nonintrusive appliance load monitoring for nonlinear devices. In *Multitopic Conference, 2007. INMIC 2007. IEEE International*, pages 1–5. IEEE, 2007. Cited on page 40.

G. Antoniou and L. Batten. E-commerce: protecting purchaser privacy to enforce trust. *Electronic commerce research*, 11(4):421–456, 2011. Cited on page 123.

R. Aparna and B. Amberker. A key management scheme for secure group communication using binomial key trees. *International Journal of Network Management*, 20(6):383–418, 2010. Cited on page 90.

M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53 (4):50–58, 2010. Cited on page 1.

M. Z. Ashrafi and S. K. Ng. Privacy-preserving e-payments using one-time payment details. *Computer Standards & Interfaces*, 31(2):321–328, 2009. Cited on page 68.

M. J. Atallah, M. Blanton, N. Fazio, and K. B. Frikken. Dynamic and efficient key management for access hierarchies. *ACM Transactions on Information and System Security (TISSEC)*, 12(3):18, 2009. Cited on page 90.

S. Barker, A. Mishra, D. Irwin, E. Cecchet, P. Shenoy, and J. Albrecht. Smart*: An open data set and tools for enabling research in sustainable homes. *SustKDD, August*, 2012. Cited on page 57.

J. Benaloh. Dense probabilistic encryption. In *Proceedings of the workshop on selected areas of cryptography*, pages 120–128, 1994. Cited on page 22.

G. R. Blakley. Safeguarding cryptographic keys. In *Managing Requirements Knowledge, International Workshop on*, pages 313–313. IEEE Computer Society, 1899. Cited on page 93.

G. E. Blelloch and B. M. Maggs. Parallel algorithms. In *Algorithms and theory of computation handbook*, pages 25–25. Chapman & Hall/CRC, 2010. Cited on page 61.

D. Boneh and R. Venkatesan. Breaking rsa may not be equivalent to factoring. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 59–71. Springer, 1998. Cited on page 24.

D. Boneh, E.-J. Goh, and K. Nissim. Evaluating 2-dnf formulas on ciphertexts. In *Theory of cryptography*, pages 325–341. Springer, 2005. Cited on pages 19, 22, and 27.

K. D. Bowers, M. Van Dijk, A. Juels, A. Oprea, and R. L. Rivest. How to tell if your cloud files are vulnerable to drive crashes. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 501–514. ACM, 2011. Cited on page 30.

D. R. Brown. Breaking rsa may be as difficult as factoring. *IACR Cryptology ePrint Archive*, 2005:380, 2005. Cited on page 24.

N. Burgess. Fast ripple-carry adders in standard-cell cmos vlsi. In *Computer Arithmetic (ARITH), 2011 20th IEEE Symposium on*, pages 103–111. IEEE, 2011. Cited on page 49.

M. Butler and R. Butler. Investigating the possibility to use differentiated authentication based on risk profiling to secure online banking. *Information & Computer Security*, 23(4):421–434, 2015. Cited on page 124.

N. Cao, C. Wang, M. Li, K. Ren, and W. Lou. Privacy-preserving multi-keyword ranked search over encrypted cloud data. *IEEE Transactions on parallel and distributed systems*, 25(1): 222–233, 2014. Cited on page 32.

D. Catalano, R. Gennaro, N. Howgrave-Graham, and P. Q. Nguyen. Paillier's cryptosystem revisited. In *Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 206–214. ACM, 2001. Cited on page 15.

A. Ceselli, E. Damiani, S. D. C. D. Vimercati, S. Jajodia, S. Paraboschi, and P. Samarati. Modeling and assessing inference exposure in encrypted databases. *ACM Transactions on Information and System Security (TISSEC)*, 8(1):119–152, 2005. Cited on page 36.

B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private information retrieval. *Journal of the ACM (JACM)*, 45(6):965–981, 1998. Cited on page 32.

A. Chowdhury, G. Breznik, K. Verdnik, and B. Prihavec. Customer identification and authentication procedure for online internet payments using mobile phone, Jan. 17 2012. US Patent 8,099,077. Cited on page 123.

R. Cramer, I. Damgård, and U. Maurer. General secure multi-party computation from any linear secret-sharing scheme. In *Advances in CryptologyEUROCRYPT 2000*, pages 316–334. Springer, 2000. Cited on page 93.

J. Crampton. Cryptographically-enforced hierarchical access control with multiple keys. *The Journal of Logic and Algebraic Programming*, 78(8):690–700, 2009. Cited on page 90.

S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Private data indexes for selective access to outsourced data. In *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society*, pages 69–80. ACM, 2011. Cited on pages 32 and 34.

P. Deng and L. Yang. A secure and privacy-preserving communication scheme for advanced metering infrastructure. In *Innovative Smart Grid Technologies (ISGT), 2012 IEEE PES*, pages 1–5. IEEE, 2012. Cited on page 51.

S. D. C. di Vimercati, S. Foresti, S. Paraboschi, G. Pelosi, and P. Samarati. Distributed shuffling for preserving access confidentiality. In *European Symposium on Research in Computer Security*, pages 628–645. Springer, 2013. Cited on pages 34 and 36.

T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Workshop on the Theory and Application of Cryptographic Techniques*, pages 10–18. Springer, 1984. Cited on page 24.

T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in Cryptology*, pages 10–18. Springer, 1985. Cited on page 22.

C.-I. Fan, S.-Y. Huang, and W. Artan. Design and implementation of privacy preserving billing protocol for smart grid. *The Journal of Supercomputing*, 66(2):841–862, 2013. Cited on page 40.

D. Ferraiolo, D. R. Kuhn, and R. Chandramouli. *Role-based access control*. Artech House, 2003. Cited on page 99.

S. D. Galbraith. Elliptic curve paillier schemes. *Journal of Cryptology*, 15(2):129–138, 2002. Cited on page 20.

J. Gallian. *Contemporary abstract algebra*. Cengage Learning, 2009. Cited on pages 17 and 20.

F. D. Garcia and B. Jacobs. Privacy-friendly energy-metering via homomorphic encryption. In *Security and Trust Management*, pages 226–238. Springer, 2011. Cited on page 51.

C. Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009a. Cited on page 27.

C. Gentry. Ibm touts encryption innovation, June 2009b. URL http://www.computerworld.com/s/article/9134823/IBM_touts_encryption_innovation. Cited on page 28.

C. Gentry. Toward basing fully homomorphic encryption on worst-case hardness. In *Advances in Cryptology–CRYPTO 2010*, pages 116–137. Springer, 2010. Cited on page 27.

GNU. The gnu multiple precision arithmetic library, Mar. 2014. URL https://gmplib.org/. Cited on pages 58, 81, and 108.

S. Goldwasser and S. Micali. Probabilistic encryption & how to play mental poker keeping secret all partial information. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 365–377. ACM, 1982. Cited on page 22.

V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 89–98. Acm, 2006. Cited on page 93.

G. Gray, K. Church, and T. Ayres. Virtual credit card processing system. *The ITB Journal*, 3(2):2, 2015. Cited on page 124.

V. L. Guillaume Hanrot. The multiple precision floating-point reliable library, Dec. 2013. URL http://www.mpfr.org/. Cited on pages 81 and 108.

M.-H. Guo, H.-T. Liaw, D.-J. Deng, and H.-C. Chao. Centralized group key management mechanism for vanet. *Security and Communication Networks*, 6(8):1035–1043, 2013. Cited on page 91.

W. Hart. Flint: Fast library for number theory, Dec. 2013. URL http://flintlib.org/. Cited on pages 58, 81, and 108.

J. L. Hennessy and D. A. Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2012. Cited on page 62.

J. Hoffstein, J. C. Pipher, J. H. Silverman, and J. H. Silverman. *An introduction to mathematical cryptography*. Springer, 2008. Cited on page 105.

H. Hu, J. Xu, X. Xu, K. Pei, B. Choi, and S. Zhou. Private search on key-value stores with hierarchical indexes. In *2014 IEEE 30th International Conference on Data Engineering*, pages 628–639. IEEE, 2014. Cited on page 35.

J. Hur and D. K. Noh. Attribute-based access control with efficient revocation in data outsourcing systems. *IEEE Transactions on Parallel and Distributed Systems*, 22(7):1214–1221, 2011. Cited on page 29.

IBM. Ibm researcher solves longstanding cryptographic challenge, June 2009. URL http://www-03.ibm.com/press/us/en/pressrelease/27840.wss. Cited on page 27.

T. Jung, X.-Y. Li, Z. Wan, and M. Wan. Privacy preserving cloud data access with multi-authorities. In *INFOCOM, 2013 Proceedings IEEE*, pages 2625–2633. IEEE, 2013. Cited on pages 32 and 33.

M. G. Kaosar, R. Paulet, and X. Yi. Fully homomorphic encryption based two-party association rule mining. *Data & Knowledge Engineering*, 76:1–15, 2012. Cited on pages 47 and 72.

M. A. Keresman III, R. Bhagavatula, C. Balasubramanian, and F. M. Sherwin. Secure and efficient payment processing system with account holder defined transaction limitations, May 1 2012. US Patent 8,170,954. Cited on page 124.

D. M. Konidala, M. H. Dwijaksara, K. Kim, D. Lee, B. Lee, D. Kim, and S. Kim. Resuscitating privacy-preserving mobile payment with customer in complete control. *Personal and Ubiquitous Computing*, 16(6):643–654, 2012. Cited on page 68.

A. K. Lenstra, H. W. Lenstra Jr, M. S. Manasse, and J. M. Pollard. The number field sieve. In *The development of the number field sieve*, pages 11–42. Springer, 1993. Cited on page 14.

H. W. Lenstra Jr. Factoring integers with elliptic curves. *Annals of mathematics*, pages 649–673, 1987. Cited on page 14.

M. Li, S. Yu, K. Ren, W. Lou, and Y. T. Hou. Toward privacy-assured and searchable cloud data storage services. *IEEE Network*, 27(4):56–62, 2013. Cited on page 29.

R. Li, Z. Xu, W. Kang, K. C. Yow, and C.-Z. Xu. Efficient multi-keyword ranked query over encrypted data in cloud computing. *Future Generation Computer Systems*, 30:179–190, 2014. Cited on page 30.

E. lyn Teske. Square-root algorithms for the discrete logarithm problem. In *Public-Key Cryptography and Computational Number Theory: Proceedings of the International Conference organized by the Stefan Banach International Mathematical Center Warsaw, Poland, September 11-15, 2000*, page 283. Walter de Gruyter, 2001. Cited on page 18.

D. Micciancio. A first glimpse of cryptography's holy grail. *Communications of the ACM*, 53 (3):96–96, 2010. Cited on page 22.

D. Micciancio and O. Regev. Lattice-based cryptography. In *Post-quantum cryptography*, pages 147–191. Springer, 2009. Cited on page 27.

P. Moreno-Sanchez, A. Kate, M. Maffei, and K. Pecina. Privacy preserving payments in credit networks. NDSS, 2015. Cited on page 68.

M. Nabeel and E. Bertino. Privacy preserving delegated access control in public clouds. *IEEE Transactions on Knowledge and Data Engineering*, 26(9):2268–2280, 2014. Cited on pages 32 and 33.

M. Nabeel, E. Bertino, M. Kantarcioglu, and B. Thuraisingham. Towards privacy preserving access control in the cloud. In *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2011 7th International Conference on*, pages 172–180. IEEE, 2011. Cited on page 33.

M. Nabeel, N. Shang, and E. Bertino. Privacy preserving policy-based content sharing in public clouds. *IEEE Transactions on Knowledge and Data Engineering*, 25(11):2602–2614, 2013. Cited on page 32.

M. Naehrig, K. Lauter, and V. Vaikuntanathan. Can homomorphic encryption be practical? In *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, pages 113–124. ACM, 2011. No cited.

M. Naor and A. Wool. Access control and signatures via quorum secret sharing. *Parallel and Distributed Systems, IEEE Transactions on*, 9(9):909–922, 1998. Cited on page 93.

S. K. Ng, J. Liang, and J. W. Cheng. Automatic appliance load signature identification by statistical clustering. 2009. Cited on page 40.

C. Örencik and E. Savaş. An efficient privacy-preserving multi-keyword search over encrypted cloud data with ranking. *Distributed and Parallel Databases*, 32(1):119–160, 2014. Cited on page 35.

P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in cryptologyEUROCRYPT99*, pages 223–238. Springer, 1999. Cited on pages 22 and 26.

M. Patwal and T. Mittal. A survey of cryptographic based security algorithms for cloud computing. *HCTL Open International Journal of Technology Innovations and Research*, 8: 2321–1814, 2014. Cited on page 2.

R. C. Paulet. Design and analysis of privacy-preserving protocols. 2013. Cited on pages 66 and 87.

R. Peralta and E. Okamoto. Faster factoring of integers of a special form. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 79(4):489–493, 1996. Cited on page 14.

H. Perl. The scarab fully homomorphic cryptography library, Nov. 2011. URL https://hcrypt.com//scarab-library/. Cited on pages 58, 81, and 108.

A. Plateaux, P. Lacharme, A. Jøsang, and C. Rosenberger. One-time biometrics for online banking and electronic payment authentication. In *Availability, Reliability, and Security in Information Systems*, pages 179–193. Springer, 2014. Cited on page 125.

Powercor. Everything you should know about smart meters. https://www.powercor.com.au/media/1426/about-smart-meters-june-2012.pdf. Accessed: 2014-06-09. Cited on pages 50 and 54.

D. Quadling. Lagrange's interpolation formula. *The Mathematical Gazette*, pages 372–375, 1966. Cited on pages 93, 96, and 101.

E. L. Quinn. Smart metering and privacy: Existing laws and competing policies. *SSRN eLibrary*, 2009. Cited on pages xi, 40, and 41.

A. Rial and G. Danezis. Privacy-preserving smart metering. In *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society*, pages 49–60. ACM, 2011. Cited on page 50.

R. L. Rivest and B. Kaliski. Rsa problem. In *Encyclopedia of cryptography and security*, pages 532–536. Springer, 2005. Cited on page 15.

R. L. Rivest, L. Adleman, and M. L. Dertouzos. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978a. Cited on pages 21 and 27.

R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978b. Cited on pages 23, 65, and 86.

D. P. Rodgers. Improvements in multiprocessor system design. In *ACM SIGARCH Computer Architecture News*, volume 13, pages 225–231. IEEE Computer Society Press, 1985. Cited on page 62.

S. Roy and P. Venkateswaran. Online payment system using steganography and visual cryptography. In *Electrical, Electronics and Computer Science (SCEECS), 2014 IEEE Students' Conference on*, pages 1–5. IEEE, 2014. Cited on page 123.

A. Ruiz-Martínez, Ó. C. Reverte, and A. F. Gómez-Skarmeta. Payment frameworks for the purchase of electronic products and services. *Computer Standards & Interfaces*, 34(1):80–92, 2012. Cited on page 123.

S. Ruj and A. Nayak. A decentralized security framework for data aggregation and access control in smart grids. *IEEE transactions on smart grid*, 4(1):196–205, 2013. Cited on page 51.

A. Rupp, F. Baldimtsi, G. Hinterwälder, and C. Paar. Cryptographic theory meets practice: Efficient and privacy-preserving payments for public transport. *ACM Transactions on Information and System Security (TISSEC)*, 17(3):10, 2015. Cited on page 68.

A. Saha and N. Manna. *Digital principles and logic design.* Jones & Bartlett Learning, 2009. Cited on page 45.

P. Samarati. Data security and privacy in the cloud. In *International Conference on Information Security Practice and Experience*, pages 28–41. Springer, 2014. Cited on page 30.

N. B. Shah, K. Rashmi, K. Ramchandran, and P. V. Kumar. Privacy-preserving and secure distributed storage codes. Globecom, 2011. Cited on page 35.

A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979. Cited on pages 86, 89, 93, 99, 101, and 104.

N. Shang, M. Nabeel, F. Paci, and E. Bertino. A privacy-preserving approach to policy-based content dissemination. In *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, pages 944–955. IEEE, 2010. Cited on page 33.

T. Shimizu, I. Hisato, and H. Sasaoka. Group secret key agreement based on radio propagation characteristics in wireless relaying systems. *IEICE transactions on communications*, 95(7): 2266–2277, 2012. Cited on page 91.

N. P. Smart and F. Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *Public Key Cryptography–PKC 2010*, pages 420–443. Springer, 2010. Cited on pages 28, 43, 45, 58, 69, 81, 92, 95, 105, 108, and 126.

W. Sun, B. Wang, N. Cao, M. Li, W. Lou, Y. T. Hou, and H. Li. Privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking. In *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*, pages 71–82. ACM, 2013. Cited on pages 30, 32, and 35.

H. Takabi, J. B. Joshi, and G.-J. Ahn. Security and privacy challenges in cloud computing environments. *IEEE Security & Privacy*, (6):24–31, 2010. Cited on page 1.

G. W.-H. Tan, K.-B. Ooi, S.-C. Chong, and T.-S. Hew. Nfc mobile credit card: the next frontier of mobile payment? *Telematics and Informatics*, 31(2):292–307, 2014. Cited on page 124.

T. Tassa. Generalized oblivious transfer by secret sharing. *Designs, Codes and Cryptography*, 58(1):11–21, 2011. Cited on page 93.

M. Van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan. Fully homomorphic encryption over the integers. In *Advances in Cryptology–EUROCRYPT 2010*, pages 24–43. Springer, 2010. Cited on page 28.

D. Wagner, B. Schneier, et al. Analysis of the ssl 3.0 protocol. In *The Second USENIX Workshop on Electronic Commerce Proceedings*, pages 29–40, 1996. Cited on page 122.

B. Wang, B. Li, and H. Li. Knox: privacy-preserving auditing for shared data with large groups in the cloud. In *International Conference on Applied Cryptography and Network Security*, pages 507–525. Springer, 2012a. Cited on pages 32 and 33.

B. Wang, B. Li, and H. Li. Oruta: Privacy-preserving public auditing for shared data in the cloud. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 295–302. IEEE, 2012b. Cited on pages 32 and 33.

C. Wang, S. S. Chow, Q. Wang, K. Ren, and W. Lou. Privacy-preserving public auditing for secure cloud storage. *IEEE Transactions on computers*, 62(2):362–375, 2013. Cited on pages 30 and 33.

H. Wang and L. V. Lakshmanan. Efficient secure query evaluation over encrypted xml databases. In *Proceedings of the 32nd international conference on Very large data bases*, pages 127–138. VLDB Endowment, 2006. Cited on page 36.

M. Weiss, A. Helfenstein, F. Mattern, and T. Staake. Leveraging smart meter data to recognize home appliances. In *Pervasive Computing and Communications (PerCom), 2012 IEEE International Conference on*, pages 190–197. IEEE, 2012. Cited on page 40.

C. K. Wong, M. Gouda, and S. S. Lam. Secure group communications using key graphs. *Networking, IEEE/ACM Transactions on*, 8(1):16–30, 2000. Cited on page 90.

Q. Wu, Y. Mu, W. Susilo, B. Qin, and J. Domingo-Ferrer. Asymmetric group key agreement. In *Advances in Cryptology-EUROCRYPT 2009*, pages 153–170. Springer, 2009. Cited on page 91.

Z. Xiao and Y. Xiao. Security and privacy in cloud computing. *IEEE Communications Surveys & Tutorials*, 15(2):843–859, 2013. Cited on page 29.

K. Yang and X. Jia. Data storage auditing service in cloud computing: challenges, methods and opportunities. *World Wide Web*, 15(4):409–428, 2012. Cited on page 32.

S. Yekhanin. Private information retrieval. *Communications of the ACM*, 53(4):68–73, 2010. Cited on page 34.

M. Zeifman and K. Roth. Nonintrusive appliance load monitoring: Review and outlook. *Consumer Electronics, IEEE Transactions on*, 57(1):76–84, 2011. Cited on page 40.

X. Zou and Y.-S. Dai. A robust and stateless self-healing group key management scheme. In *Communication Technology, 2006. ICCT'06. International Conference on*, pages 1–4. IEEE, 2006. Cited on page 91.